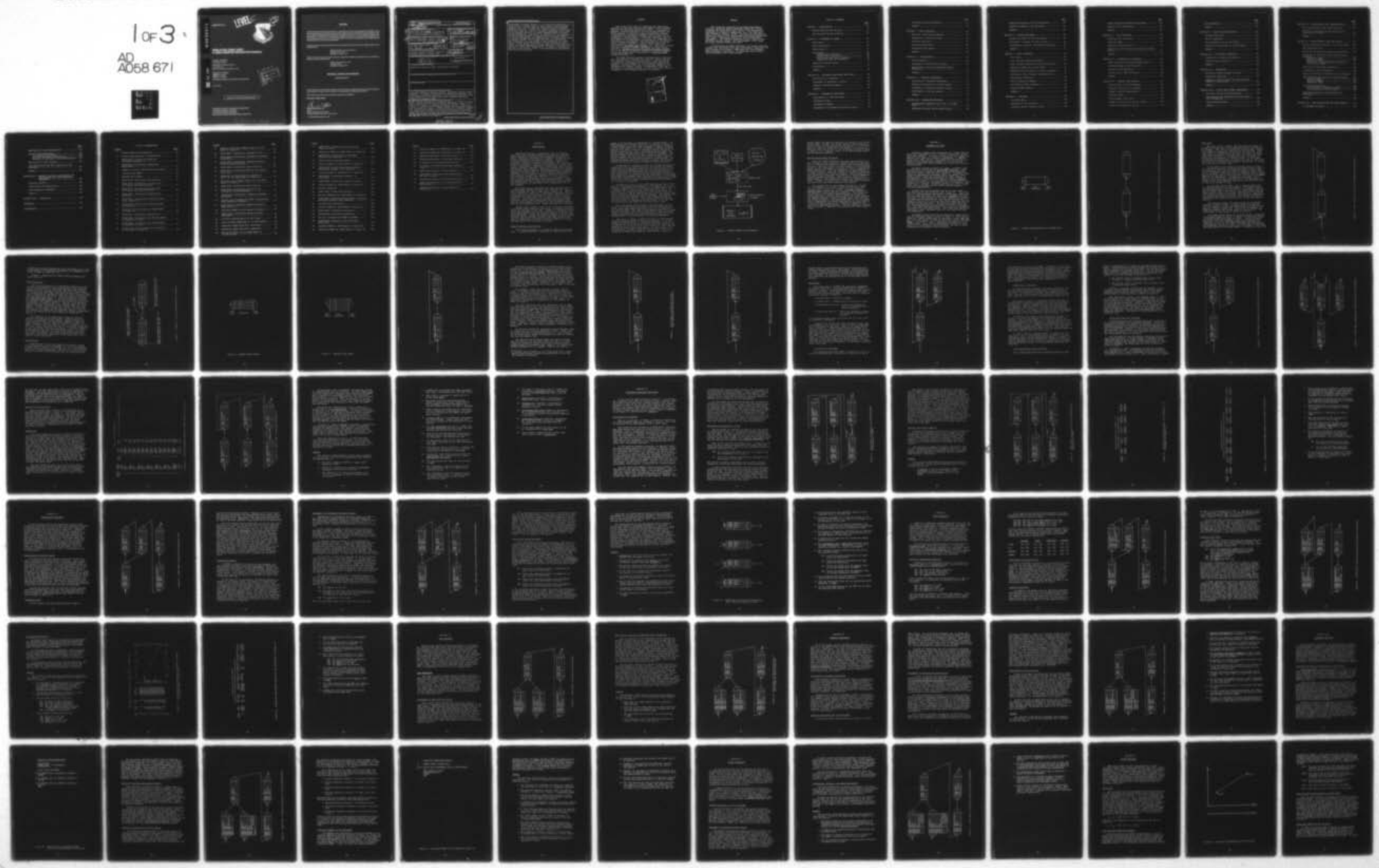
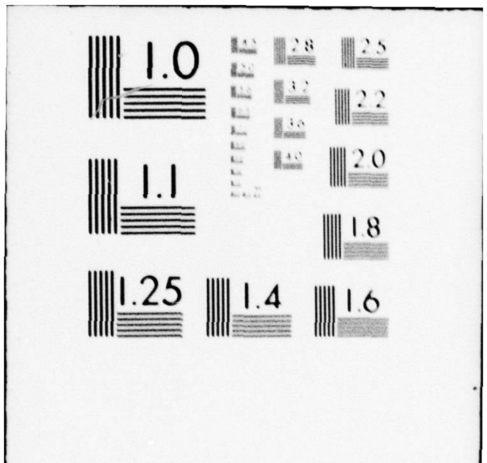
AD-A058 671 PRITSKER AND ASSOCIATES INC WEST LAFAYETTE IND F/G 9/2
SIMULATION USING SAINT: A USER-ORIENTED INSTRUCTION MANUAL.(U)
JUL 78 D B WORTMAN, S D DUKET, D J SEIFERT F33615-76-C-5012
UNCLASSIFIED AMRL-TR-77-61 NL

1 OF 3
AD
A058 671




1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

B.S. 2

AMRL-TR-77-61

SIMULATION USING SAINT: A USER-ORIENTED INSTRUCTION MANUAL

DAVID B. WORTMAN
STEVEN D. DUKET
PRITSKER & ASSOCIATES, INC.
P. O. BOX 2413
WEST LAFAYETTE, INDIANA 47906

DEBORAH J. SEIFERT
REUBEN L. HANN
GERALD P. CHUBB
AEROSPACE MEDICAL RESEARCH LABORATORY

JULY 1978

Approved for public release; distribution unlimited.

AEROSPACE MEDICAL RESEARCH LABORATORY
AEROSPACE MEDICAL DIVISION
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

NOTICES

TECHNICAL REVIEW AND APPROVAL

AMRL-TR-77-61

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

[illegible]

CHARLES BATES, JR.
Chief
Human Engineering Division
Aerospace Medical Research Laboratory

AIR FORCE/56780/28 August 1978 — 500

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

18

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER 17 AMRL-TR-77-61	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 SIMULATION USING SAINT: A USER-ORIENTED INSTRUCTION MANUAL •		5. TYPE OF REPORT & PERIOD COVERED 9 Final Report, August 1975-December 1976
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) 10 David B. Wortman, Steven D. Duket, Deborah J. Seifert, Reuben L. Hann, Gerald P. Chubb		8. CONTRACT OR GRANT NUMBER(s) 15 F33615-76-C-5012
9. PERFORMING ORGANIZATION NAME AND ADDRESS Pritsker & Associates* P.O. Box 2413 West Lafayette, IN 47906; Aerospace Med. Research Lab.** WPAFB, OH 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62202F; 16 7184; 718413; 71841303 17 13
11. CONTROLLING OFFICE NAME AND ADDRESS Aerospace Medical Research Laboratory Aerospace Medical Division, AFSC Wright-Patterson Air Force Base, OH 45433		12. REPORT DATE 11 July 1978
		13. NUMBER OF PAGES 199
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) 12 199p.		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computers; Modeling; Operator Loading; Operations Research; Mission Analysis; Survivability/Vulnerability; Man-Machine Systems; Crew Performance; Networks; Simulation; Discrete Event Simulation; Next Event Simulation; Continuous Simulation; Combined Simulation; Simulation Languages; SAINT

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

SAINT (Systems Analysis of Integrated Networks of Tasks) is a network modeling and simulation technique developed to assist in the design and analysis of complex man-machine systems. SAINT consists of a symbol set for modeling systems and a computer program for analyzing such models. SAINT provides the conceptual framework for representing systems that consist of discrete task elements, continuous state variables, and interactions between them. While SAINT was designed for modeling manned systems in which human

over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

1B

78 09 13 032

391382

performance is a major concern, it is potentially applicable to a broad class of systems--those in which discrete and continuous elements are to be portrayed and quantified and whose behavior exhibits time-varying properties. SAINT provides a mechanism for describing these dynamics so a systematic assessment can be made of the relative contribution system components make to overall system performance. This report is intended as an introduction to the basic SAINT concepts for persons with no previous exposure to the technique. This is accomplished by introducing small groups of related concepts and then applying them to a hypothetical man-machine system. This system is used throughout the report and is modeled in increasing detail as new concepts are introduced. In this manner the reader can follow the development of a relatively complex man-machine system model while acquiring knowledge of the full SAINT capabilities. A summary is provided at the end of each section to aid in reviewing material presented in that portion of the text.

SUMMARY

This report describes the symbolism and terminology required for modeling systems using SAINT. SAINT (Systems Analysis of Integrated Networks of Tasks) is a package of computer routines designed to aid the system designer and human engineer in analyzing complex man-machine systems. It provides the conceptual framework which allows the development of system models in which men, machines, and the environment are represented. It permits the assessment of the effect of the component characteristics of the system on overall system performance. The procedures for using the SAINT simulation program to analyze system models are described in The SAINT User's Manual (1). The overall structure and individual FORTRAN subprograms of SAINT are described in Documentation for the SAINT Simulation Program (20). The use of an external statistical analysis package to analyze SAINT output is described in Analyzing SAINT Output Using SPSS (3).

All SAINT modeling concepts are described in this manual. The manual is divided into sections, where each section details a set of related concepts. Each new concept is presented, discussed, and then illustrated through an example. A single example of a man-machine system is used throughout the manual in order to demonstrate SAINT capabilities in a systems context. At the end of each section, a summary of the concepts presented is provided.

PREFACE

This study was initiated by the Human Engineering Division, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio 45433. The research was conducted by Pritsker & Associates, Inc., P.O. Box 2413, West Lafayette, Indiana 47906. The work was performed in support of Project 7184 "Man-Machine Integration Technology," task 718413 "Man-Machine Models for System Performance Assessment." The research sponsored by this contract was performed between August, 1975 and December, 1976, under Air Force contract F33615-76-C-5012.

In developing the SAINT technique, the authors received guidance and assistance from many individuals, especially the staff members of Pritsker & Associates and the Aerospace Medical Research Laboratory. We appreciate their efforts and acknowledge their contributions.

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

Figure | Page

Figure Page

SECTION I

INTRODUCTION

Communications play a fundamental role in research, especially when interdisciplinary activities are involved. Mental images and concepts are satisfactory as long as a single researcher is working on a problem. As soon as two or more individuals are working together, a vehicle for expressing ideas and concepts is necessary. The use of networks or graphs as communication vehicles for researchers is well established. Examples of networks are circuit diagrams, free-body diagrams, signal flow graphs, block diagrams, and PERT and GERT networks. Networks are models of systems. These models may be used for both communication and analysis purposes. In many cases, it is the former purpose which is significant as it permits a concise, explicit definition of the pertinent concepts the researcher wishes to convey. Once a network is developed, effort can be concentrated on analyzing the system by analyzing the network model. When the network model can be used for both descriptive and analysis procedures, the researcher has a significant tool at his disposal.

The human factors specialist has long advocated the use of operational sequence diagrams (4), function flow logic block diagrams (5), and to some extent, models (6). Besides Siegel and Wolf's early use of simulation techniques to evaluate operators performing discrete tasks, there are other precedents for using simulation to portray manual tracking tasks (7). What appeared to be required was an integrated framework that not only assimilated and consolidated these previous achievements but allowed for the systematic enrichment of such technical tools. SAINT provides this framework.

SAINT is not a model; there are no "built-in" parameters or distributions, although a wide range of standard distributions are available to the user. SAINT is a framework within which any proposed model may be described and exercised; the only requirement is that the model be capable of being quantitatively stated. Changes can easily be made as the situation dictates and the network diagram of the model enables the user to communicate to others the basic concepts of the system model he or she is trying to develop.

Network Modeling and Analysis

The SAINT philosophy is to separate modeling from analysis. A graphic approach to modeling is taken in which the system to be analyzed is represented by a network model. The SAINT approach to problem resolution is depicted in Figure 1. A SAINT network model describes a system in network terms using the SAINT symbol set. The fundamental elements of SAINT networks are tasks, resources (equipment and/or personnel) required to perform the tasks, relationships among tasks, and system status variables referred to as state variables. System performance is related to which tasks are performed, the manner in which they are realized, and the extent to which certain states of the system are achieved or maintained. The generalized SAINT symbol set provides a vehicle for modeling resources performing tasks to accomplish system objectives.

In addition to providing a fixed set of symbols which are integrated to form a network model of the system, the SAINT network modeling approach allows for the specification of the conditions and constraints under which the system operates. These conditions and constraints may include such factors as time constraints on resources and the environmental conditions under which the resources must perform the tasks. By providing the means for specifying such conditions and constraints, SAINT allows the analyst to depict man-machine system performance in a variety of situations.

An important factor in the SAINT modeling approach is the analyst's experience and knowledge of the system. It is the analyst's repsonsibility to integrate the SAINT symbol set into a network model of the system, identifying each task to be performed, the resources required to perform the tasks, the state variables, and the conditions that initiate interactions between state variables and task performance. The analyst also specifies the environmental conditions under which the system is required to operate. Thus, while SAINT provides the necessary tools for developing system models, it is the analyst who develops the models by utilizing those tools.

Once the analyst has developed a network model of the system, the SAINT simulation program will automatically generate system performance estimates, that is, an analysis of the network model. The input to the SAINT program is an alphanumeric representation of the network model. The simulation program processes the input and performs a simulation of the network model to obtain estimates of system performance.

The level of detail at which a system or system segment should be modeled cannot be specified _a priori_. It is the analyst's repsonsibility to determine the level of detail to be included in the network model based upon the nature of the problem he is trying to solve and an analysis of the task components and their interrelationships. He must decide if it is sufficient to model a task as a single unit, or if it is necessary to model each component individually. Detailed

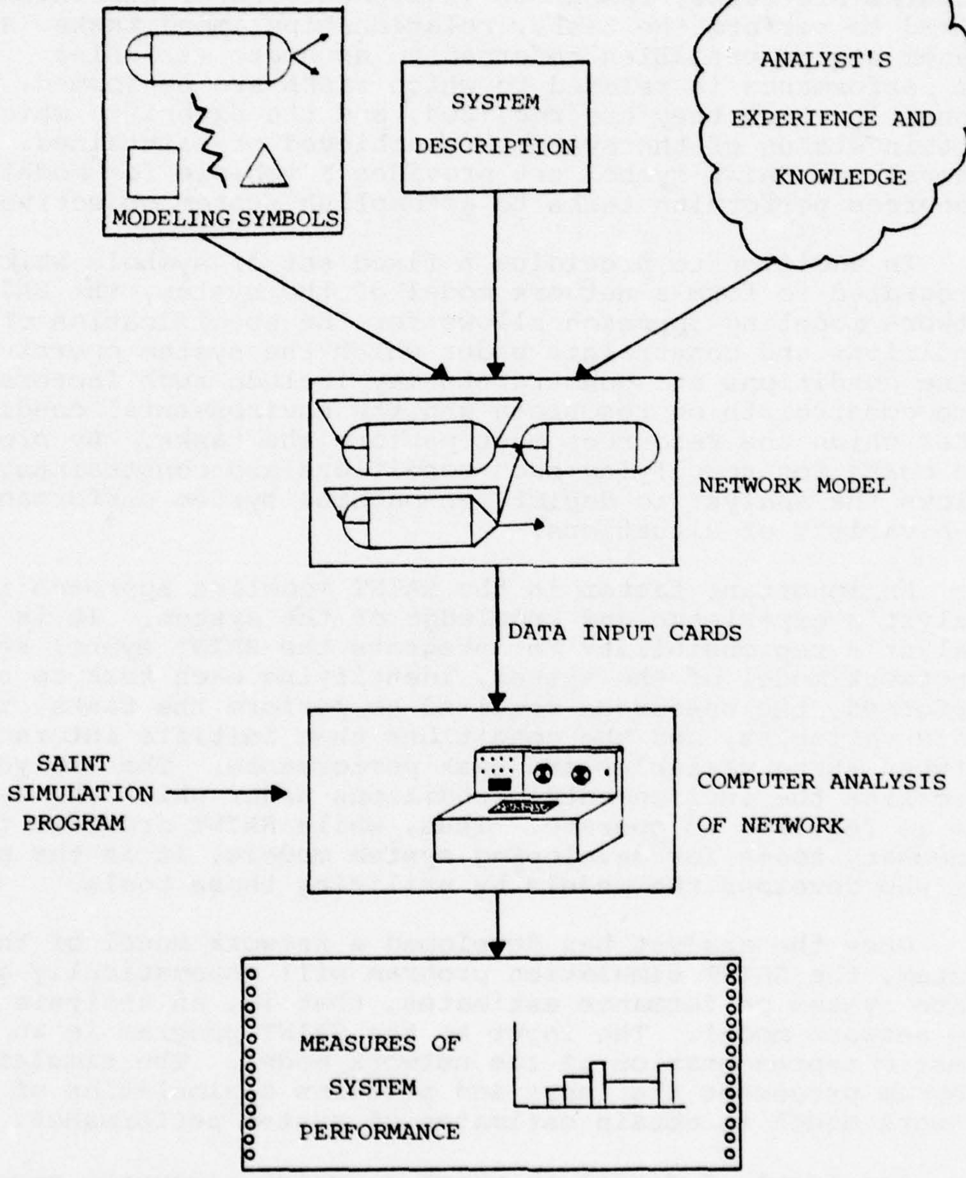

MODELING SYMBOLS
SYSTEM DESCRIPTION
ANALYST'S EXPERIENCE AND KNOWLEDGE
NETWORK MODEL
DATA INPUT CARDS
SAINT SIMULATION PROGRAM
COMPUTER ANALYSIS OF NETWORK
MEASURES OF SYSTEM PERFORMANCE

Figure 1. Network Modeling and Analysis.

network models of specific tasks may be included in an overall system model. Further, the output derived from simulating a detailed model may be used as the description of a particular task in an overall system mode. The concept of hierarchical modeling, where the output of one simulation is used as input to another, is an inherent aspect of the SAINT network modeling and analysis approach.

User-Oriented Manual Structure

Since the SAINT simulation technique was designed to be used to model a wide variety of complex systems, it incorporates an extensive and sometimes complex set of modeling capabilities. For this reason, a single example of a man-machine system is used throughout this manual in order to demonstrate SAINT capabilities, from the fundamental to the advanced, in a systems context. Each new concept is presented, discussed, and then illustrated through an example. Each succeeding section presents new concepts, building on the more fundamental concepts already presented. This structure allows you to learn SAINT at a pace commensurate with your desires and abilities. Further, it allows you to master only those concepts that meet your individual modeling requirements.

The example used in this manual pertains to a system that is composed of a group of operators monitoring and controlling the flight of a number of ground controlled vehicles (GCV) through the use of visual (CRT) displays of the vehicles' flight paths and parameters. The overall approach will be to verbally describe possible alternative representations of the GCV system, define the SAINT concepts and symbols required in order to model them, and to present the resulting models. The representations of the GCV system used to demonstrate the SAINT capabilities are only intended to be illustrative and are not intended to represent any existing or hypothesized system. The GCV example is intended only to provide a framework for presenting the SAINT capabilities.

SECTION II

SEQUENCES OF TASKS

Consider a simple system consisting of a single operator whose sole responsibility is to launch a single GCV (ground controlled vehicle). In order to achieve the GCV launch, the operator must press two buttons in succession: a "LAUNCH" button and a "GCV" button. The pressing of the buttons are tasks to be performed by the operator. Let us begin by defining a few basic SAINT concepts that relate to this man-machine system.

The basic element of SAINT is a task. In SAINT, tasks are related to one another by precedence relations. A precedence relation stipulates that a task can be initiated only after another task has been completed. Precedence relations are represented by branches (arcs, connecting lines) between tasks. Tasks are represented by nodes (Figure 2). The combination of branches and nodes is called a network. A network is a model: it is an abstract representation of a system, just like a set of equations can be an abstract representation of a system. Networks, however, are easier to conceive, and as we shall see for SAINT, network analysis is provided automatically.

A SAINT task has associated with it an input side, a task description, and an output side (Figure 2). The input side of a task specifies the number of predecessor tasks that must be completed before the task can be released. The term "released" is used instead of "started" because all predecessor tasks can be completed but the task not started due to a resource conflict (i.e., two or more tasks requiring the same resource at the same time). The source of these conflicts will be discussed more fully in later sections of this manual. The task description consists of a series of parameters associated with task performance. The output side represents the means by which successor tasks will be identified upon task completion.

The SAINT network model of the simple GCV system involving two sequential tasks is shown in Figure 3. Each task included in a SAINT network must be assigned a unique task number. The task number is shown on the output side of the task symbol. In Figure 3, task 1 represents the pressing of the LAUNCH button. This task is immediately followed, as indicated by the precedence relation (branch), by task 2, which represents the pressing of the GCV button.

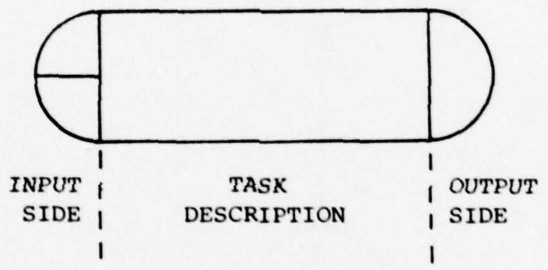


Figure 2. General Representation of a SAINT Task.

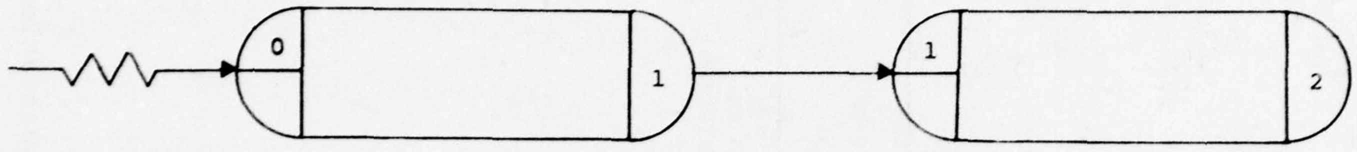


Figure 3. SAINT Model Illustrating Tasks and Precedence Relations.

Task Input

The input side of a SAINT task specifies the number of predecessor tasks that must be completed before the task can be released. The task is released for starting when a specified number of predecessor requirements is satisfied. Since tasks might be released more than once, and since the first time usually represents a special case, the design of SAINT allows for two values to be associated with the number of predecessor requirements to release a task: 1) the number of predecessor requirements to release the task for the first time; and 2) the number of predecessor requirements to release the task after the first time.

In the model of the simple GCV system shown in Figure 3, the task which represents pressing the LAUNCH button (task 1) is released at the time the simulation begins. Thus, there are no predecessor tasks that must be completed before task 1 can be released. This type of task is referred to as a source task and is indicated by the wavy input line to the left-hand side of the task symbol. In addition, the number "0" is placed in the upper left-hand portion of the task symbol to indicate that no predecessor tasks need to be completed prior to the first release of task 1.

In the same network model, task 2 requires that task 1 be completed before it can be released. The LAUNCH button must be pushed before the GCV button. A precedence relation exists between task 1 and task 2. The branch drawn from the output side of task 1 to the input side of task 2 represents the required precedence relation. The number of predecessor task completions required for the first release of task 2 is specified in the upper left-hand corner of the task symbol and is 1.

Now consider another GCV system in which the operator launches a series of GCVs. In this system, the operator will be required to press the LAUNCH button and then the GCV button to launch a single GCV. Once a GCV is launched, the identical process will be performed for the next GCV. The SAINT model of this system is shown in Figure 4.

The SAINT network model for this GCV system is identical to the model shown in Figure 3, except that there now exists a feedback branch from task 2 to task 1. After both task 1 and task 2 have been completed, task 1 is released again. Since task 1 is to be released immediately following the completion of task 2, the number of predecessor requirements for subsequent release of task 1 is one as shown in the lower left-hand portion of the task symbol. In addition, since task

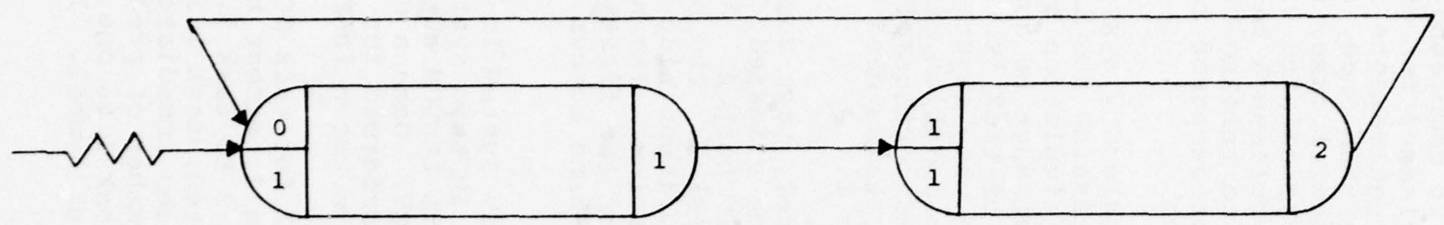


Figure 4. SAINT Model Illustrating Predecessor Requirements.

2 should be released immediately after the completion of task 1, the number of predecessor requirements for subsequent release of task 2 is also set to 1.

Figure 5 summarizes the SAINT modeling concepts presented thus far.

Task Description

The interior portion of a task symbol contains all task description information, such as performance time characteristics, statistics to be collected, and other task related quantities to be discussed in later sections of this manual. It is subdivided into rows, with each row containing a specific type of descriptive information about the task. Further, each row is divided into two parts. The left-hand part contains the task description code. It is used to identify the type of information that appears in the right-hand part of the row. The standard task symbol contains four rows for descriptive information, as shown in Figure 6. However, only the information necessary to describe a task need be shown on a task symbol. If fewer than four rows are needed, the remaining rows can be left blank; if more than the four rows provided are required, we simply add the necessary number of additional rows to the bottom of the task symbol, as illustrated in Figure 7.

For example, let us assume that we want to verbally identify the tasks of our GCV system. SAINT allows us to specify an alphanumeric description or label for each task in the network. The task description code for a label is LABL. Assuming we want to refer to task 1 as "LAUNCH" and to task 2 as "GCV", the SAINT model of the GCV system illustrated in Figure 4 would appear as in Figure 8. To be more complete in our description, we might have labeled task 1 as "Press LAUNCH Button", and task 2 as "Press GCV Button", but these are longer expressions. The SAINT computer programs will only store up to eight characters of the label, so it is desirable to keep them short.

Task Duration

Task duration, or the time required to perform a task, is also specified on the task symbol. It can be a constant, a sample from a probability distribution, or a value obtained from a user-written subprogram. The use of user-written subprograms to define task performance will be explained later.

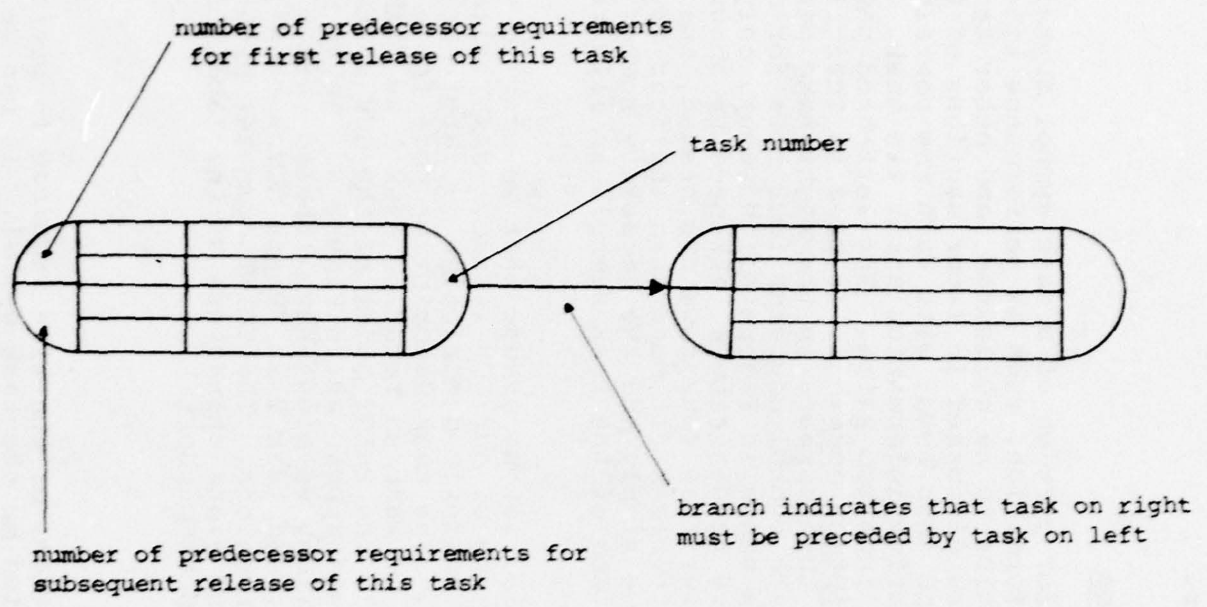

number of predecessor requirements for first release of this task
task number
branch indicates that task on right must be preceded by task on left
number of predecessor requirements for subsequent release of this task

Figure 5. Summary of Initial SAINT Modeling Concepts.

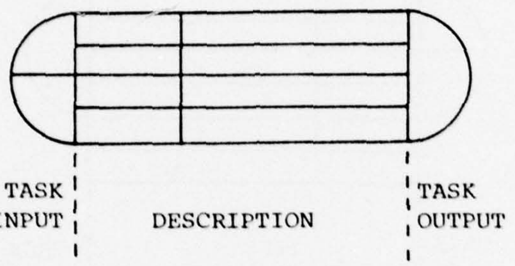


Figure 6. Standard Task Symbol.

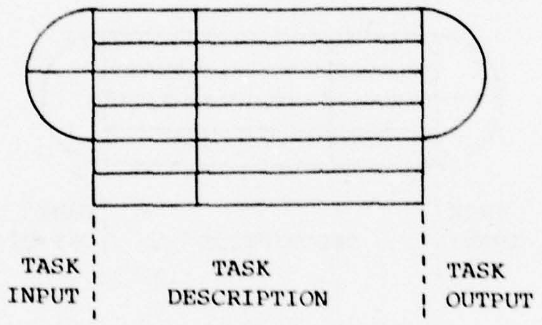


Figure 7. Expanded Task Symbol.

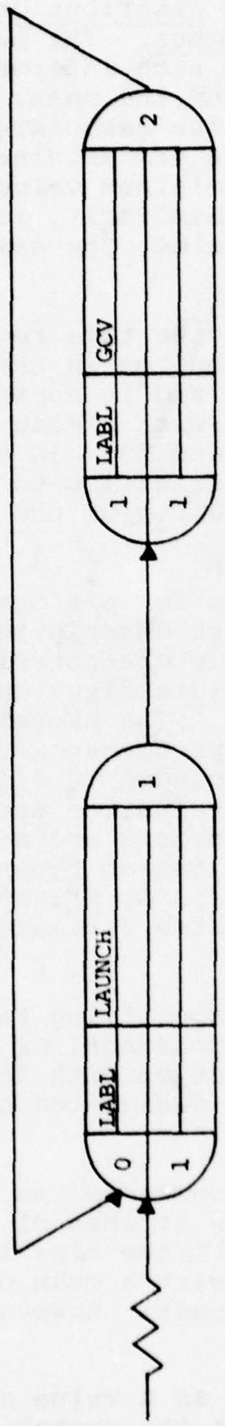


Figure 8. SAINT Model Illustrating Task Labeling.

The SAINT simulation program can produce samples from probability distributions, utilizing the information provided by the user in the associated distribution set, identified through a distribution set number. The parameters included in a distribution set provide such information as the distribution type (e.g., normal), and the mean, standard deviation, minimum value, and maximum value associated with the probability distribution. The samples are obtained by SAINT such that if a sample is less than the minimum value, the sample value is given the minimum value. Similarly, if the sample value is greater than the maximum value, the sample value is assigned the maximum value.

For example, assume that the time required to press either the LAUNCH or the GCV button in the model shown in Figure 4 has been determined and is appropriately described as being normally distributed with a mean of 1.0 second and a standard deviation of 0.5 seconds. In order to assign these task performance time characterisitics to both tasks 1 and 2, additional information is required on the task symbol, as shown in Figure 9.

The task description code for performance time is TIME. The right-hand side of the task description row, which represents the task performance time characteristics, now contains the alphanumeric symbol "DS" (Distribution Set) and the number "1", separated by a comma. The presence of these characters indicates that the task performance time of each task is defined by the information included in distribution set 1. On input, we would define distribution set 1 as a normal distribution with a mean of 1.0 second and a standard deviation of 0.5 seconds. Eleven distribution types are available in SAINT: constant[1], normal, uniform, Erlang, lognormal, Poisson, beta, gamma, beta fitted to three estimates, triangular, and Weibull.

An alternate method for specifying task performance times is through the use of Scaled Constant, or "SC". SAINT takes the value specified in conjunction with "SC", and divides it by the scale factor (which is designated on input) to compute the task performance time.

Now consider the situation where the task performance time of task 2 is not the same as that of task 1. In this situation, modeled in Figure 10, the time to perform task 1 remains normally distributed with a mean of 1.0 seconds and a standard deviation of 0.5 seconds. However, the time to

[1] A constant can be thought of as a value drawn from a probability distribution in which the probability is one that the constant is obtained.

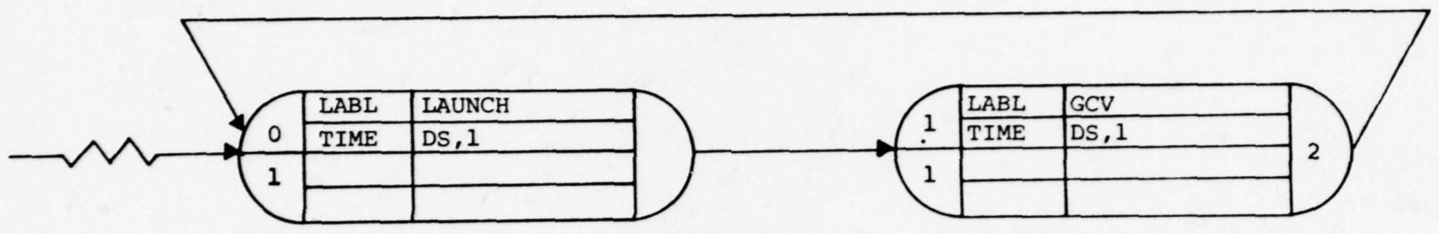


Figure 9. SAINT Model Illustrating A Single Task Performance Time Specification.

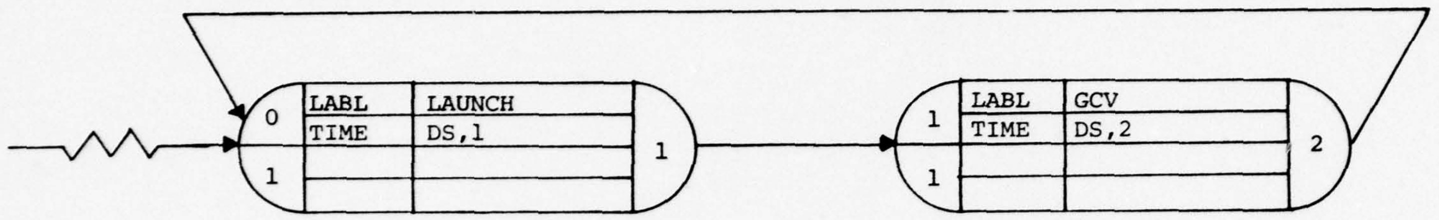


Figure 10. SAINT Model Illustrating Multiple Task Performance Time Specifications.

perform task 2 is now characterized by the information contained in distribution set 2. For example, this distribution set might include information describing a uniform distribution with a minimum value of 0.5 seconds and a maximum value of 1.5 seconds. In this manner, we can specify task performance times that are governed by any of the available distributions.

Task Output

The output side of a SAINT task represents a branching or decision operation. Following completion of a task, a selection is made as to which branches emanating from the task should be selected. The branching type dictates the method by which this selection is made. The four types of branching operations included in SAINT are:

1. Deterministic - select all branches
2. Probabilistic - probabilistically select one branch
3. Conditional-take first - select first branch for which the specified condition is satisfied
4. Conditional-take all - select all branches for which the specified conditions are satisfied

If no branches emanate from the output side of a task, then no branching is performed.

A special case is the task whose completion could cause the stopping of the simulation. This is called a sink task, and is designated by the symbol shown on the output side of task 3 (Figure 11). SAINT allows multiple sink tasks to be included in a single network model and the completion of more than one sink task, or more than one completion of a single sink task, may be required for the simulation to be stopped. The number of sink task completions required is specified on input. A sink task can have the same performance time characteristics as any other task in the SAINT network. Thus, a sink task has all the capabilities associated with a regular task with the exception of branching. No branching is allowed from a sink task.

Deterministic Branching

The branching operations shown in Figures 1, 3, 4, 8, 9, and 10 are deterministic, indicated by a semi-circle (D) on

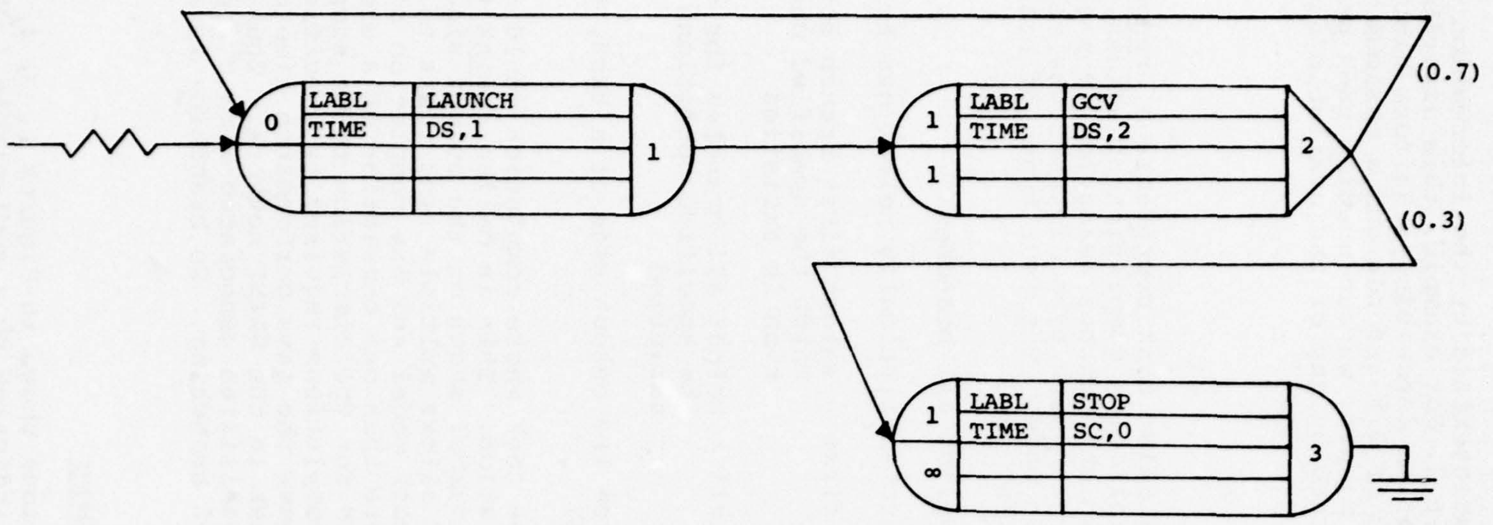


Figure 11. SAINT Model Illustrating Probabilistic Branching.

the output side of each task symbol. Variations in the shape of the output side of each task will be used to reflect branching options and will be discussed on subsequent pages. When a deterministic branching operation is specified, all branches emanating from the task are selected upon task completion. Thus, the number of requirements for all successor tasks connected to the completed task by the branches is reduced by one. Essentially, each branch has a probability of 1.0 of being selected.

Probabilistic Branching

For probabilistic branching, each branch emanating from the task has an associated probability of being selected. Only one of the branches is selected upon completion of the task. The sum of the probabilities associated with the branches emanating from a task with probabilistic output must be 1.

Once again, consider the situation modeled in Figure 10. In that model, there was a probability of 1.0 (deterministic branching) of launching another GCV once the preceding GCV was launched. However, assume now that after the operator has launched a GCV, there is a 70% chance of his launching another GCV and a 30% chance of another GCV not being launched. In the latter case, the simulation will be halted. The SAINT network model of this new system is shown in Figure 11.

In Figure 11, the shape of the output side of task 2 is altered to reflect the probabilistic branching operation required. The triangle shape (▷) is used to indicate probabilistic branching. The probability of performing task 1 upon the completion of task 2 (launching an additional GCV) is 0.7. The probability of not launching an additional GCV (proceeding to task 3 upon completion of task 2) is 0.3. Note that the sum of the probabilities associated with the branches emanating from task 2 sum to 1.0 (0.7 + 0.3 = 1.0).

If task 3, which is labeled "STOP", is selected as the successor to task 2, the simulation will be halted. The only function of task 3 in the network model shown in Figure 11 is to indicate the end of the simulation. As such, it takes no time to "perform". Thus, the time to perform task 3 is specified as 0 through the use of the performance time specification "SC".

Conditional-Take First Branching

For a conditional-take first branching operation, each branch is specified with a conditon, and the branches are ordered. Each condition is tested sequentially in the order in which it appears on the data input cards, and the first branch whose condition is satisfied is selected. All of the conditions that are recognized by SAINT will be presented later. However, two of the available conditions are:

1. The current value of simulated time is less than or equal to a specified value (TLV), and

2. The current value of simulated time is greater than a specified value (TGV).

Consider a situation in which the GCV operator is not allowed to launch another GCV whenever the current value of simulated time is greater than 10 time units. The SAINT network model of this situation is shown in Figure 12.

Upon completion of task 2, the feedback branch to task 1 is selected if the current value of simulated time is less than or equal to 10. When this branch is selected, the operator will launch another GCV. However, if the current value of simulated time is greater than 10, the branch to task 3 will be selected, no additional GCVs will be launched, and the simulation will end. In Figure 12, the output side of task 2 is modified to reflect the conditional-take first branching operation. The symbol for conditional-take first branching is ▷.

Conditional-Take All Branching

A conditional-take all branching operation is similar to the conditional-take first branching operation. Any of the conditions recognized by SAINT for the conditional-take first branching operation are available for use with conditional-take all branching. However, in the case of conditional-take all branching, the condition on every branch emanating from the task is evaluated. For every condition that is satisfied, the corresponding branch is selected.

Let us assume for our GCV example that after the operator stops launching GCVs, 1) he must press a "FINISHED" button indicating that he is no longer launching GCVs, and 2) the last GCV launched must fly for 10 time units before the simulation is stopped. The SAINT model for this situation is given in Figure 13.

In Figure 13, task 3 represents the operator pressing the "FINISHED" button. Its performance time is governed by distribution set 3. Task 4 represents the flight of the last GCV (assume the scale factor is 1). After the operator launches

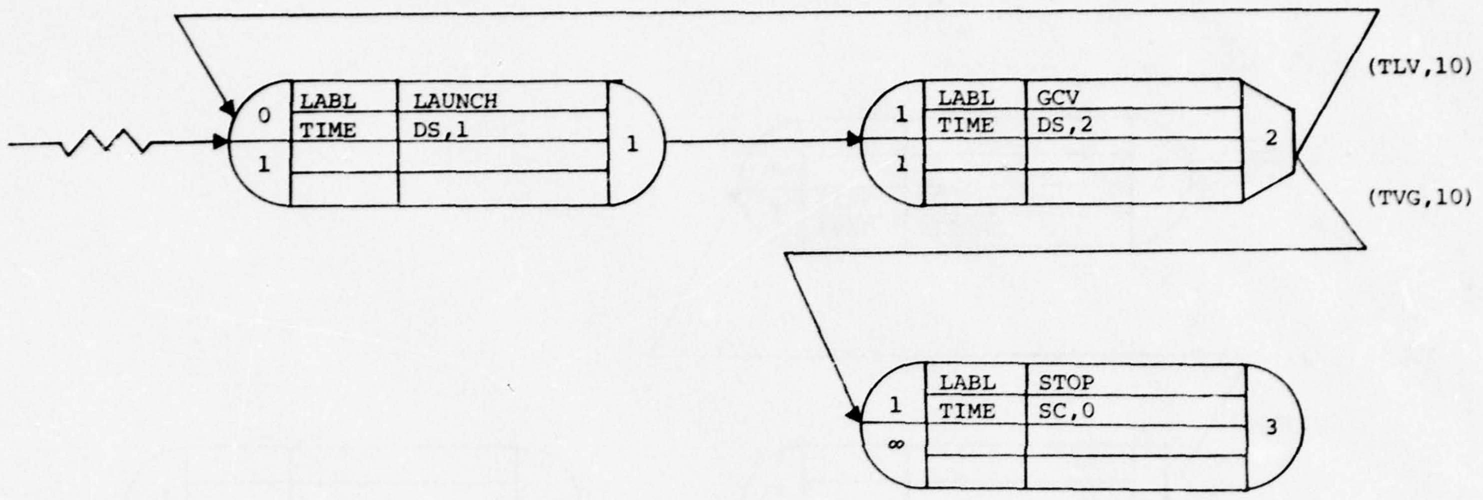


Figure 12. SAINT Model Illustrating Conditional-Take First Branching.

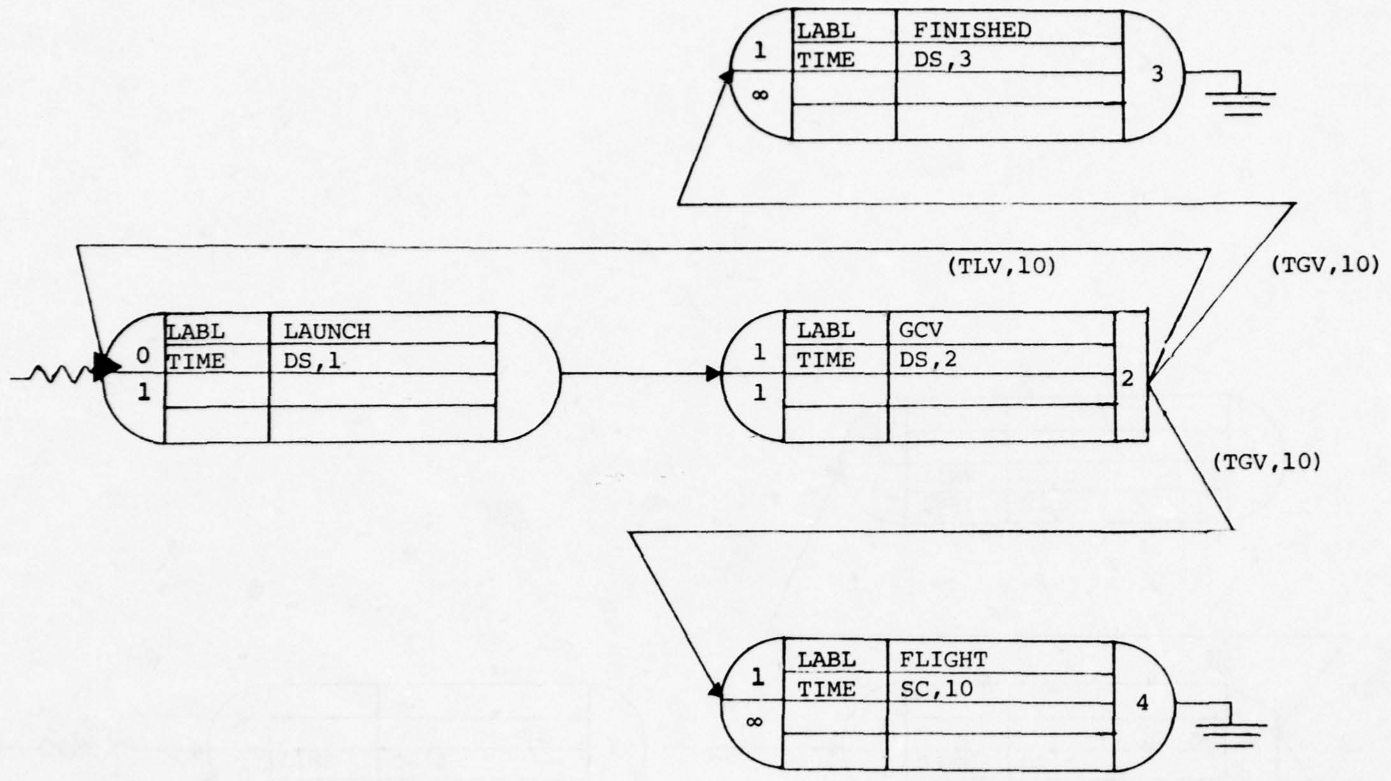


Figure 13. SAINT Model Illustrating Conditional-Take All Branching.

the last GCV, we want both tasks 3 and 4 to be released simultaneously. Thus, the output side of task 2 is squared off (□) to reflect the conditional-take all branching that allows branching for all conditions that are satisfied. Also, we want both tasks 3 and 4 to be completed before the simulation is complete. For this reason, both tasks are sink tasks and we specify (through input) that 2 sink task completions are required for the simulation to be completed.

Detailed Iteration Report

Among the SAINT user options is the provision for a detailed iteration report. Figure 14 illustrates this option for the SAINT model in Figure 13. The report identifies the release, start, and completion times for each task performed as the simulation progresses. In addition, the priority and resources assigned to the task are listed (these concepts will be introduced in later sections). This output option is especially useful in uncovering errors in model logic and in communicating the details of model operation to others.

Aggregation

Let us now assume that we wish to model a situation in which the GCV operator will launch a single GCV and then monitor its flight during the remainder of the simulation. In order to launch a GCV, the operator needs only to press the "LAUNCH" and "GCV" buttons. However, to monitor the GCV, that is, to obtain the deviation of the GCV from its desired flight path at any point in time, the operator must request computer action to compute the GCV's flight deviation. To do this, the operator must press the "COMPUTER" button to interact with the computer and then press "STATUS" to inform the computer that he desires the GCV's deviation status. The computer will then obtain and process the desired information and report the current time and the GCV deviation to the CRT console. The operator writes this information down on the appropriate recording form. Additional requests for deviation status require the operator to repeat the above tasks.

The exact manner by which the GCV deviation values are computed and the SAINT concepts used to represent the computation and display of information will be subjects of later sections of this manual. However, a plausible SAINT representation of this situation is shown in Figure 15.

```
                                   **DETAILED OUTPUT OF ITERATION NUMBER    1**

TASK OR   TASK OR
MONITOR   MONITOR          TASK       EVENT        TIME OF
NUMBER     LABEL         PRIORITY      TYPE         EVENT    ---------------------RESOURCES ASSIGNED TO TASK--------------------

   1      LAUNCH               0     RELEASE           0
   1      LAUNCH               0     START             0

   1      LAUNCH                   * COMPLETE        .78
   2      GCV                  0     RELEASE         .78
   2      GCV                  0     START           .78

   2      GCV                      * COMPLETE       2.52
   1      LAUNCH               0     RELEASE        2.52
   1      LAUNCH               0     START          2.52

   1      LAUNCH                   * COMPLETE       3.02
   2      GCV                  0     RELEASE        3.02
   2      GCV                  0     START          3.02

   2      GCV                      * COMPLETE       3.69
   1      LAUNCH               0     RELEASE        3.69
   1      LAUNCH               0     START          3.69

   1      LAUNCH                   * COMPLETE       4.47
   2      GCV                  0     RELEASE        4.47
   2      GCV                  0     START          4.47

   2      GCV                      * COMPLETE       5.73
   1      LAUNCH               0     RELEASE        5.73
   1      LAUNCH               0     START          5.73

   1      LAUNCH                   * COMPLETE       6.86
   2      GCV                  0     RELEASE        6.86
   2      GCV                  0     START          6.86

   2      GCV                      * COMPLETE       7.36
   1      LAUNCH               0     RELEASE        7.36
   1      LAUNCH               0     START          7.36

   1      LAUNCH                   * COMPLETE       8.42
   2      GCV                  0     RELEASE        8.42
   2      GCV                  0     START          8.42

   2      GCV                      * COMPLETE       9.32
   1      LAUNCH               0     RELEASE        9.32
   1      LAUNCH               0     START          9.32

   1      LAUNCH                   * COMPLETE       9.91
   2      GCV                  0     RELEASE        9.91
   2      GCV                  0     START          9.91

   2      GCV                      * COMPLETE      11.03
   3      FINISHED             0     RELEASE       11.03
   4      FLIGHT               0     RELEASE       11.03
   3      FINISHED             0     START         11.03
   4      FLIGHT               0     START         11.03

   3      FINISHED                 * COMPLETE      11.53

   4      FLIGHT                   * COMPLETE      21.03
```

Figure 14. Detailed Iteration Report for SAINT Model in Figure 13.

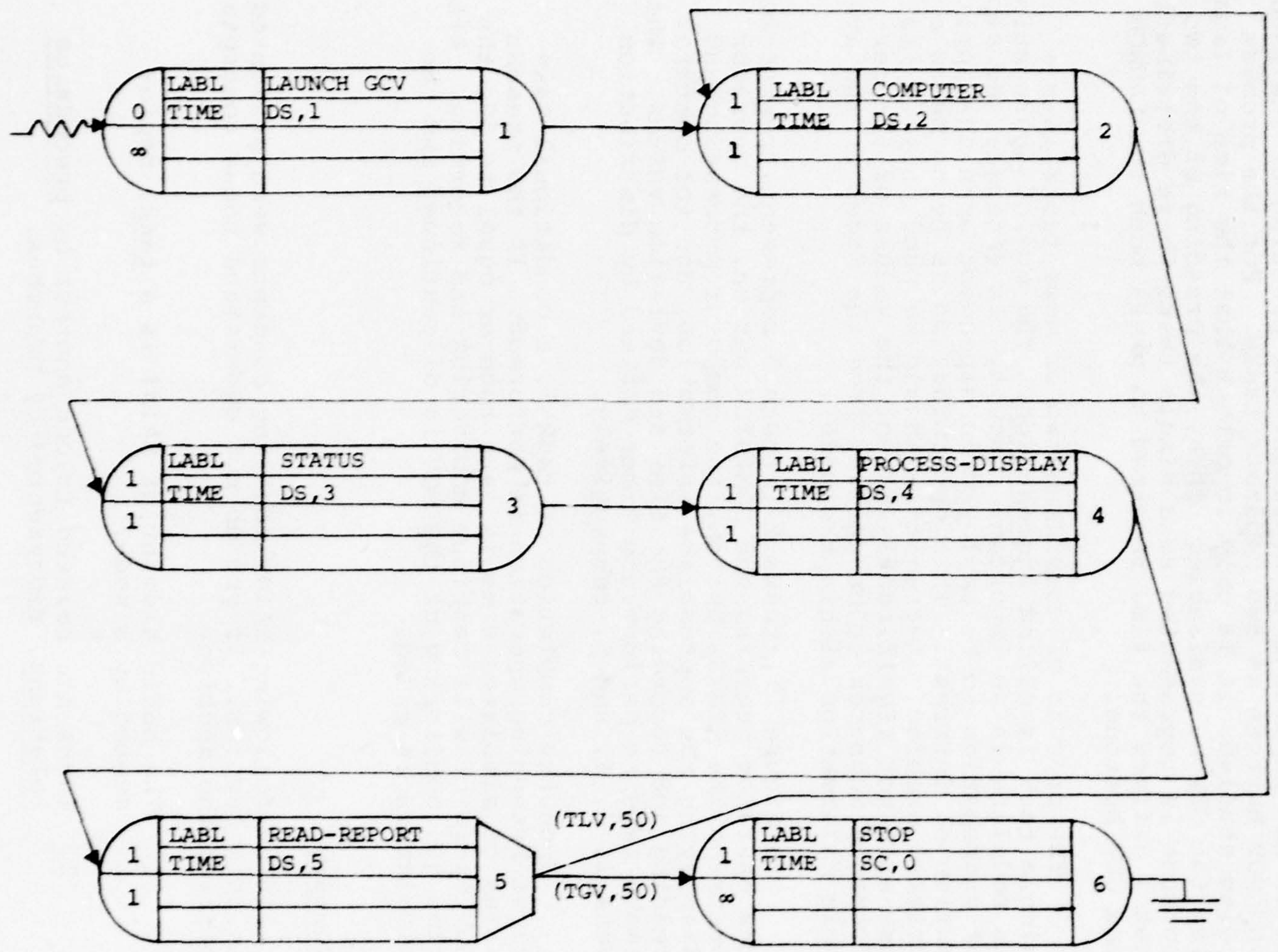


Figure 15. SAINT Model Illustrating Aggregation.

In this model, task 1 represents the pressing of both the LAUNCH button and the GCV button. We have assumed that it is not important to represent the pressing of the LAUNCH and GCV buttons as two separate tasks. For the problem being studied, it is only important that the time of launching the GCV be available. Thus, the pressing of the two buttons is represented as a single task, where distribution set 1 defines the time required to press both the LAUNCH and GCV buttons.

The process of combining two or more tasks into a single task is called aggregation. The use of aggregation in modeling is an important concept. In general, the amount of aggregation to be employed is dependent upon the application of interest. This determination is by no means a trivial problem. Aggregation should be used only if its use will not significantly affect the values of the performance measures to be derived from the model or the available information within the model.

In Figure 15, tasks 2 through 5 represent, respectively, the operator pressing the COMPUTER button, the operator pressing the STATUS button, the computer processing and displaying the appropriate information, and the operator reading and recording the time and deviation values. These tasks require performance times defined by distribution sets 2, 3, 4, and 5, respectively.

Upon the completion of task 5, a conditional-take first branching operation is performed. If the present value of simulated time is less than or equal to 50, the GCV operator will continue monitoring and recording. After time 50, monitoring of the GCV is discontinued and the simulation is ended.

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, re-read the section.

1. The basic element of SAINT is a task, represented by a node.

2. Tasks are related to one another by precedence relations, represented by branches.

3. The combination of tasks and precedence relations is a network, an abstract representation of a system.

4. A SAINT task is divided into three sections: task input, task description, and task output.

5. Each task in a network is identified by a unique task number.

6. The input side of a task specifies the number of predecessor tasks, identified by the precedence relations (branches) that must be completed before the task is released.

7. SAINT requires two values for the specification of predecessor requirements: the number for first release and the number for subsequent release.

8. A source task has no predecessor requirements for first release. Source tasks are automatically released at the beginning of a simulation.

9. The task description portion of a SAINT task contains parameters associated with task performance and statistics collection.

10. Each row of the task description contains a specific type of information, identified by an alphanumeric task description code.

11. An alphanumeric label can be specified for each task, identified by the task description code LABL.

12. Task duration can be a constant, a sample from a probability distribution, or a value obtained from a user-written subprogram.

13. Probability distributions are described in distribution sets, identified by a unique distribution set number.

14. The task description code for task duration is TIME.

15. The alphanumeric symbol DS indicates that task performance time is a sample from a distribution set.

16. The alphanumeric symbol SC indicates that task performance time is a scaled constant (a constant divided by a scale factor specified on input).

17. The shape of the output side of a SAINT task represents a branching operation, the means by which branches emanating from a task are selected.

18. Deterministic branching, represented by a semi-circle (D), selects all branches.

19. Probabilistic branching, represented by a triangle (▷), selects a single branch probabilistically.

20. Conditional-take first branching, represented by a truncated triangle (⏢), selects the first branch for which the specified condition is satisfied.

21. Conditional-take all branching, represented by a rectangle (▯), selects all branches for which the specified conditions are satisfied.

22. A task whose completion could cause the end of a simulation is a sink task.

23. The process of combining two or more tasks into a single task is aggregation.

SECTION III

RESOURCES ASSOCIATED WITH TASKS

In the last GCV model presented (Figure 15), an operator and a computer were required to perform tasks. However, since all tasks in that model were released sequentially, the performance of tasks by the operator and computer was dictated solely by the task release requirements specified. Thus, the operator and computer were implicit components of the model. In order to make the operator and computer explicit components of the model, we identify them as resources.

Definition of a Resource

What is a resource? In SAINT, a resource is defined as any non-consumable commodity (equipment, operator, etc.) that is required for the performance of one or more tasks.

When resources are included in a SAINT model, there are two requirements which must be satisfied before a task can be performed. First, as indicated in the previous section, a specified number of predecessor tasks must be completed before the task is released. Second, the resources required to perform the task must be available. If the resources are available, then they are set to work performing the task. If the resources are not available, i.e., they are busy performing other tasks, the scheduling of task performance must be delayed until they become available.

In the previous examples presented, the operator could have been modeled explicitly as a resource. However, because no tasks were competing for this operator and because we had not introduced the concept of a resource, the operator was not defined as a resource. With the introduction of resources, we can illustrate the procedures for including resources in a network description. An important advantage of using resources in one's model is the automatic calculation of the utilization of each resource by SAINT. However, for each task in the network, we must decide which resources are required for performance (each resource is identified by a unique number).

For the latest GCV example, define resource number 1 as the computer and resource number 2 as the operator. From Figure 15, we can see that task 1 (the operator launching the GCV) will require only the operator for its performance. Similarly, task 2 (the operator requesting the computer) requires only the operator for its performance. However, task

3 (inputting the status request) requires the interaction of both the operator and the computer. Task 4 (processing the GCV deviation) is performed solely by the computer. Task 5 (recording the GCV deviation) requires only the operator. Lastly, task 6, which ends simulation, requires neither the operator nor the computer for its performance.

For this model we could have decided that we wanted to compute the utilization of the pencil that the operator uses in recording a GCV deviation or the keyboard through which the operator makes requests. However, one need only define as resources those items which may cause delays in the scheduling of task perfromance due to resource unavailability or those items whose utilization is of significant interest. The selection of the resources to be included in a model is based on the analyst's description of the system and the art of knowing what is important. These decisions are based in part on the intended use of the model. The capability to make good decisions identifies the experienced analyst.

Assignment of Resources to Tasks

The assignment of resources to tasks for our GCV system is presented in Figure 16. The task description portion of the task symbol is used to specify the resources associated with the task. The task description code for resources associated with the task is RESR. In the right-hand portion of the resource specification appears the resource requirement code for the task and the resource associated with the task.

What is the resource requirement code? It specifies the method by which SAINT assigns resources to tasks after the tasks have been released. There are two possible methods for assigning resources to tasks:

AND All resources specified (if any) are required for the performance of the task;

OR One of the resources specified is required for the performance of the task.

The method of resource assignment that we select directly affects the conditions under which the task is performed.

An AND requirement stipulates that all of the resources associated with the task must be available before the task can be started. To illustrate this concept, again consider Figure 16. The operator (resource 2) and the computer (resource 1) are assigned to the task labeled STATUS (node 3) by prescribing the AND resource requirement code. Since we know which specific resources are to perform which specific tasks, we are able to use the AND method.

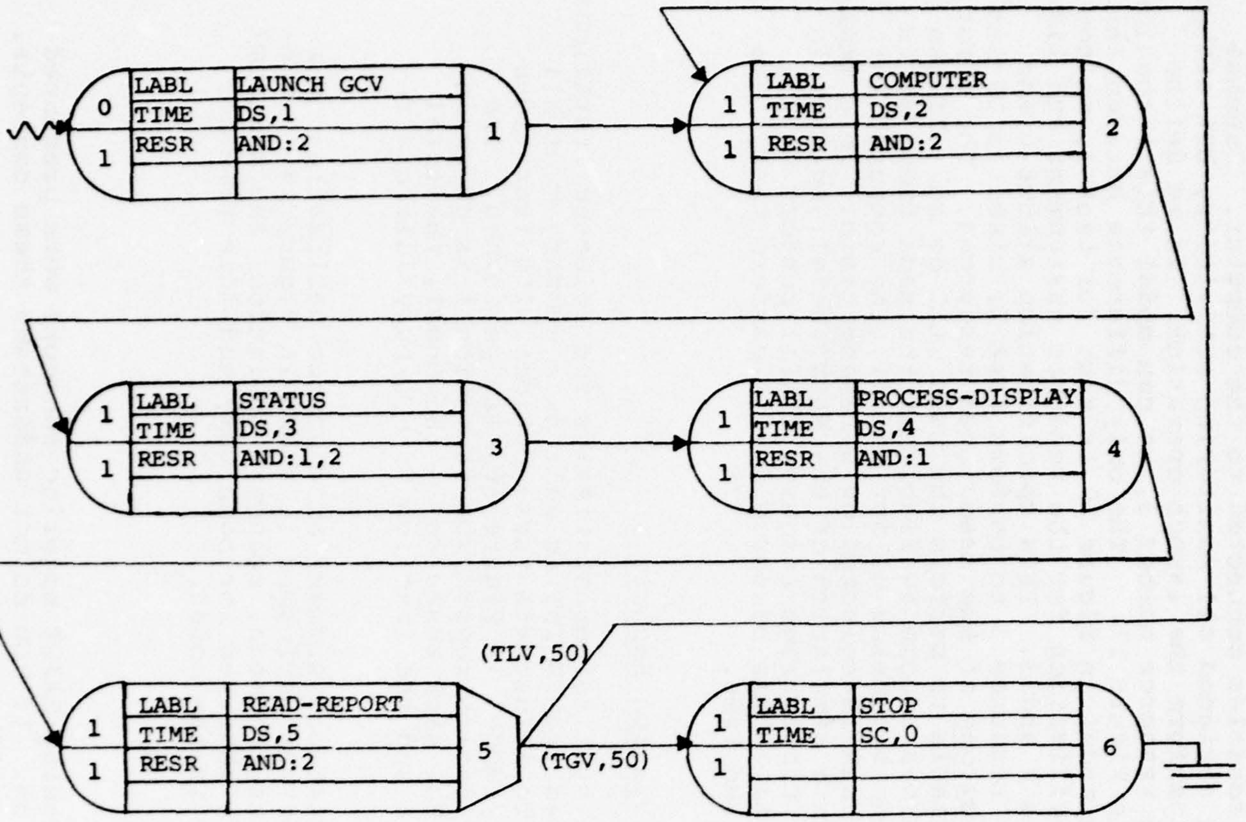


Figure 16. SAINT Model Illustrating the AND Assignment of Resources to Tasks.

Now consider the situation involving two operators in the GCV system. Both of these operators are situated in front of CRT consoles connected to the computer. Suppose operator 1 is assigned all monitoring activities, but either operator can perform the launch operation. If we define operator 2 as resource number 3, we can model this situation as depicted in Figure 17. The only difference between this model and the model in Figure 16 is that for task 1, we now use an OR specification for the resource assignment and list both resources 2 and 3. This specification allows either resource 2 or resource 3 to perform task 1, since an OR task is performed by one of the associated resources. The first resource available to perform the task will do so. If two or more resources become available at the same time (as is the case at the beginning of our model), the resource that performs the task is selected on a random basis. This implies that if we run a simulation of the SAINT model depicted in Figure 17, on the average resource 2 will perform task 1 50% of the time, while resource 3 will perform task 1 the other 50% of the time.

Resource Utilization Reports

Resource utilization statistics for selected iterations can be obtained from SAINT as well as a summary over all iterations. To illustrate these options, 100 iterations of the network model in Figure 16 were performed. The resource utilization report for iteration 1 is shown in Figure 18. For each resource in the model, the total time busy/idle and the fraction of time busy/idle are reported.

The statistical summary of resource utilization over the 100 iterations performed is shown in Figure 19. The report includes the mean, standard deviation, and minimum/maximum values observed for both busy and idle time for each resource in the model.

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, re-read this section.

1. A resource is any non-consumable commodity (equipment, operator, etc.) that is required for the performance of one or more tasks.

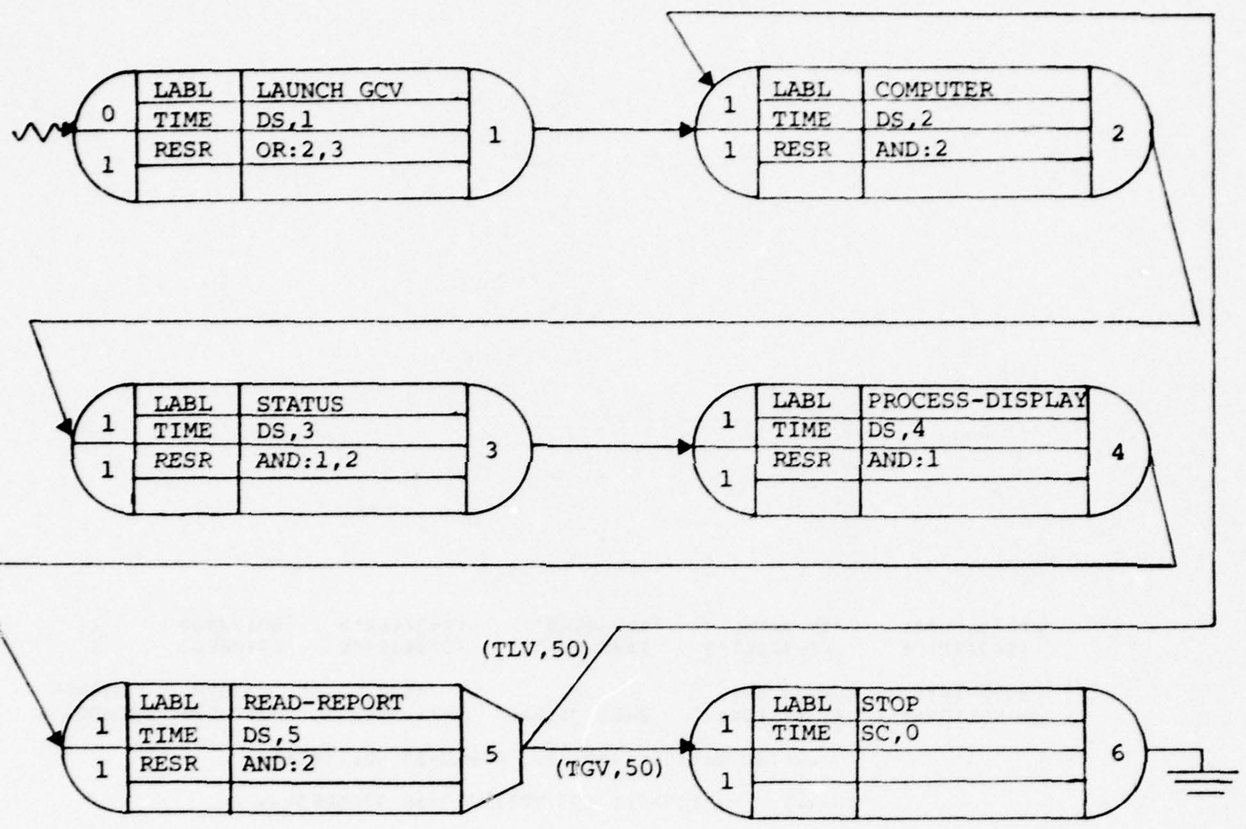


Figure 17. SAINT Model Illustrating the OR Assignment of Resources to Tasks.

```
                  **RESOURCE UTILIZATION FOR ITERATION    1**

                  *ITERATION LENGTH =   6.0414E+01 TIME UNITS*

RESOURCE   RESOURCE    TCTAL TIME     TOTAL TIME     FRACTION CF     FRACTION CF
 NUMBER     LABEL         EUSY           IDLE         TIME BUSY       TIME IDLE

   1       COMPUTER    3.1375E+01     2.9039E+01     5.1933E-01      4.8067E-01
   2       OPERATOR    4.1933E+01     1.8482E+01     6.9408E-01      3.0592E-01
```

Figure 18. Resource Utilization Report for Iteration 1 of the Model in Figure 16.

```
                        ***RESCURCE UTILIZATION SUMMARY REPORT***

                  *FRACTIONS OF TIME BUSY AND ICLE COLLECTED FOR  100 ITERATICNS*

RESOURCE  RESOURCE    --------------FRACTION OF TIME BUSY--------------    --------------FRACTION CF TIME IDLE--------------
 NUMEER    LABEL        AVERAGE      STD DEV      MINIMUM      MAXIMUM       AVERAGE      STD DEV      MINIMUM      MAXIMUM

    1     COMPUTER    5.6591E-01   4.3108E-02   4.8816E-01   6.7889E-01    4.3409E-01   4.3108E-02   5.1184E-01   3.2111E-01
    2     OPERATOR    6.4779E-01   4.1819E-02   5.4390E-01   7.6684E-01    3.5221E-01   4.1819E-02   4.5610E-01   2.3316E-01
```

Figure 19. Resource Utilization Summary Report for the Model in Figure 16.

2. When resources are included in a SAINT model, a second requirement must be satisfied before the task can be performed. After the task is released, all required resources must be available (not performing other tasks).

3. If the required resources are not available, the scheduling of task performance is delayed until they do become available.

4. SAINT automatically collects and reports the utilization of each resource included in a model.

5. Each resource is identified by a unique number.

6. The task description code for resources associated with a task is RESR.

7. The right-hand side of the RESR row contains the resource requirement code and a list of resources, identified by number, associated with the task.

8. The resource requirement code specifies the method used by SAINT to assign the associated resources to a task after it has been released. The two possible methods are:

AND	All resources specified (if any) are required for task performance.
OR	One of the resources specified is required for task performance.

9. If the OR specification is selected and two or more resources are available, the assignment of a resource to perform the task is made on a random basis.

SECTION IV

INFORMATION ATTRIBUTES

In the previous section, we constructed a model which represents the launch of a GCV by one or two operators followed by the monitoring and reporting of the GCV flgiht status by a single operator until mission time reaches 50. Instead of continuously monitoring the flight of one GCV, suppose we launch a series of two GCVs and determine the deviation from their flight paths after they have flown approximately 50 time units. Operator 2 launches the two GCVs and operator 1 monitors and records the deviation of these GCVs after they have flown for at least 50 time units. In order to keep our network manageable, assume that the computer is activated automatically whenever operator 1 performs the monitoring task. In this case, it is no longer necessary for us to treat the computer as a resource of the system. For convenience, we redefine resources 1 and 2 as operators 1 and 2, respectively.

Description of a New GCV Model

The model for this new system is depicted in Figure 20. Task 1 represents the launch of the first GCV. Task 2 represents the launch of the second GCV. To perform these tasks, only operator 2 is required. Both task performance times are governed by distribution set 1. Task 3 represents the 50 time unit delay before operator 1 first requests the status of a GCV. Thus, its performance time is 50 time units. Neither resource 1 nor resource 2 are required to perform task 3. Task 4 represents an aggregation of operator 1 requesting the status of the GCV, the computer computing and displaying the deviation of the GCV, and the operator recording this information. Here, we have made the assumption that the operator is required to wait at the terminal for the display and cannot perform other operations during this time. The time to perform this task is now defined by distribution set 2. Task 5 causes the simulation to end after the status of both GCVs has been recorded.

Because we wish both GCVs to have their status recorded, we do not perform task 5 until both GCVs have had operator 1 monitor and record their deviation. To accomplish this, we set the number of predecessor completions required for the first release of task 5 to 2.

Information Flow

On the surface, the model described above seems to

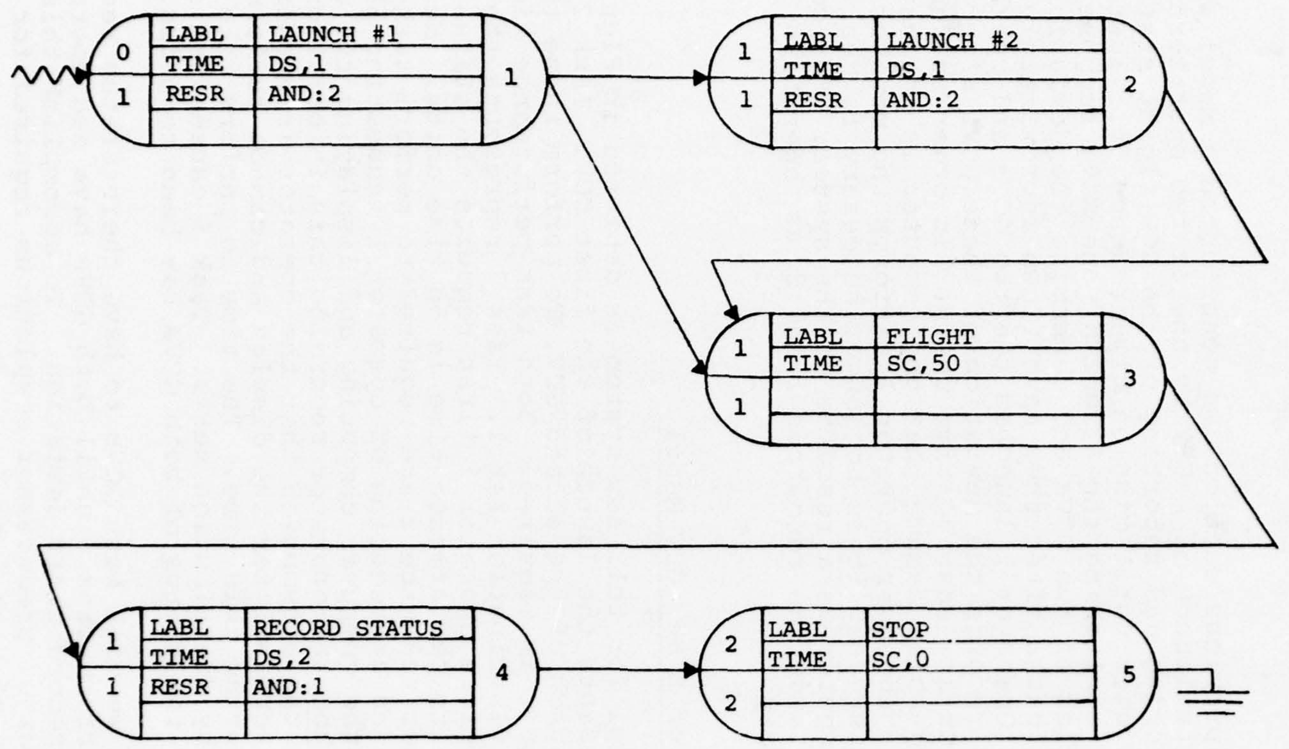


Figure 20. SAINT Model Illustrating Information Flow.

represent the situation desired. However, there is no indication in the network that the two performances of tasks 3 and 4 are different in any way, even though they are being performed for different GCVs. Essentially, the significant question is: At task 4, how does operator 1 know which GCV he is considering?

From the networks we have described, we can see that there appears to be a flow in the network dictated by the branching from task to task. This leads us to ask: Is there anything flowing between tasks, and if so, what is it? The answer to this question is that information is flowing between tasks. The information flow in all the previous examples has only been used to identify successors to the task just completed. However, in this case, we can see that we may want to pass additional information along the branches, that is, we want to inform the SAINT model which task is to be performed next and for which GCV. Specifically, we would like to tell the operator at task 4 which GCV is to be monitored. Why? Suppose that we have a way (which we will explain later) of modeling GCV flight using their equations of motion. In that case, when operator 1 at task 4 determines and records the status of the GCV, it will be necessary for the model to determine which GCV is being monitored so that the appropriate flight equations can be used to compute the deviation value.

Information Packets

In SAINT, information is organized into packets, with each information packet containing attributes. Attributes characterize items flowing through the network, signals being processed by the network , or any other concept relating to network flow. As an example, consider the GCVs being modeled in our system. Attributes such as wing span and thrust characteristics could be included in the information packet associated with each GCV.

At the start of the simulation, information packets are created at all source tasks. The number of attributes contained in each packet is specified on input. When a task is completed, the information packet associated with that task is transmitted along each precedence branch selected by the branching operation. For example, if branching is deterministic, the information packet will be sent along all branches emanating from the completed task. If a task has probabilistic output, then the information packet will only be sent along the single branch selected. If the task calls for conditional branching, then the information packet will be sent along all branches selected. This process continues until all required sink tasks have been completed.

Assignment of Information Attribute Values

Assignments of information attribute values are made at the tasks of the network through an assignment mechanism. This assignment mechanism allows us to assign a constant value or a value sampled from a probability distribution to any attribute of the information packet that is currently at the task.

In our example, suppose we wish to define information attribute 1 as the GCV number. At task 1, which represents the launch of GCV 1, we would assign information attribute 1 the value of 1. This value will then be associated with the information packet for the remainder of the simulation (unless it is changed at a subsequent task).

In the model shown in Figure 21, two deterministic branches emanate from task 1. Thus, two information packets will be created when task 1 is completed. Both will have information attribute 1 equal to 1. This first packet will proceed to task 2, indicating that the first GCV has been launched and that the second GCV can now be launched. The second packet will proceed to task 3 so that the GCV described by information attribute 1 (1) can start its 50 time unit flight.

At the start of task 2, we reassign the value of information attribute 1 in the arriving packet. It is assigned a value of 2 to indicate that this packet will now be associated with the second GCV. This reassignment is performed only for the <u>one</u> packet that arrived at task 2. Therefore, we now have two distinct information packets in our model, one representing each GCV. The packet representing GCV 2 will flow from task 2 to task 3 after the GCV is launched (task 2 is completed). The assignment mechanism that we employ for the above process is depicted in Figure 21.

The task description code for an assignment made at a task is <u>ATAS</u> (<u>at</u>tribute <u>as</u>signment). In the right-hand side of this assignment specification appears the description of the assignment. Assignments may be made at any of three points in time relating to the task (those in Figure 21 were selected arbitrarily):

REL The <u>rel</u>ease of the task

STA The <u>sta</u>rt of the task. Note that the task may be released and not started due to the unavailability of resources at the time of release.

COM The <u>com</u>pletion of the task.

Only one assignment point can be specified for each task.

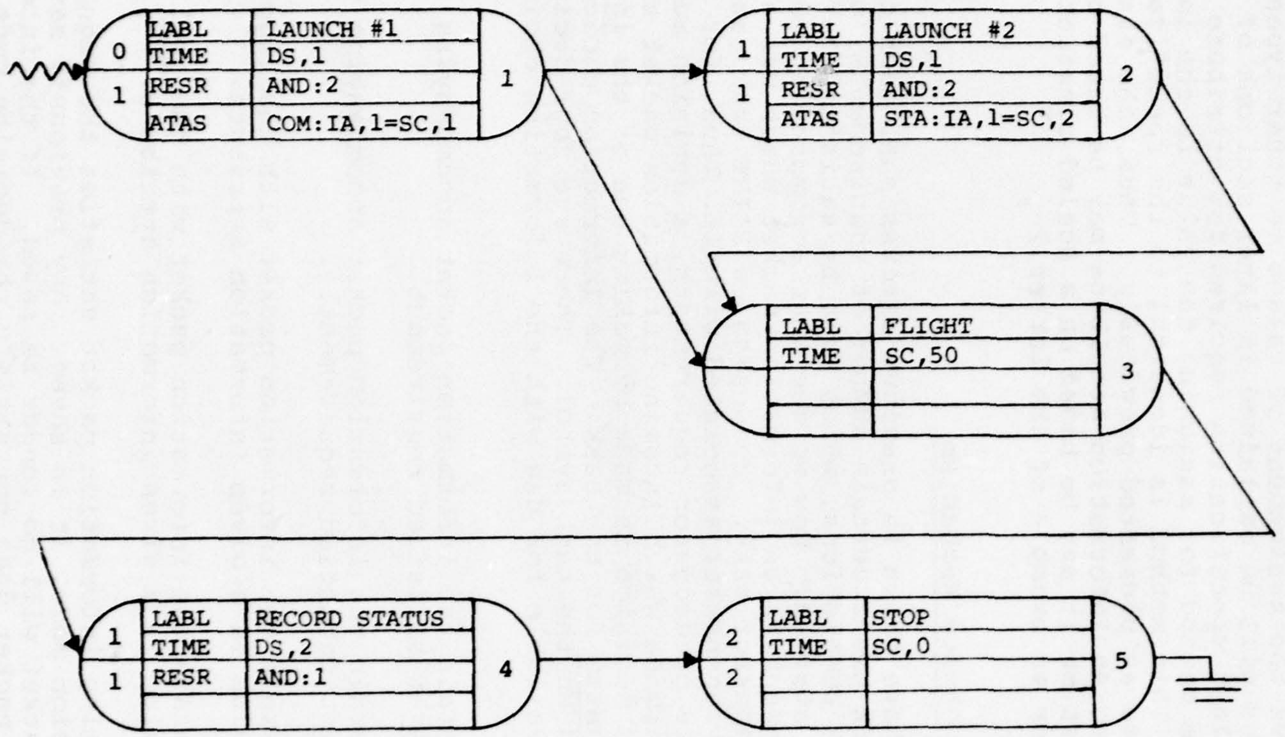


Figure 21. SAINT Model Illustrating Information Attribute Assignments.

The second specification required for an attribute assignment is the attribute type. In this case, since we are interested in information attributes, the specification is IA followed by the attribute number that will be assigned a value. (SAINT allows the assignment of values to other types of attributes. These will be explained in later sections of this manual.) The last specification required for attribute assignments is the method for assigning the value to the information attribute. This method is identical to the specification for the task time as presented previously. Thus, the assignment of a value to an information attribute may be based on a distribution set or it may be based on a scaled constant (See Figure 21 for an example of the latter.)

Information Choice Mechanism

As we have shown in previous discussions, each task in a SAINT network has a certain number of requirements, or predecessor task completions, which must be satisfied before the task can be started. Now we see that accompanying each satisfied requirement is an information packet which was associated with a predecessor task. By design, a SAINT task can only have one information packet associated with it. Thus, for tasks with multiple predecessor requirements, a decision must be made to determine which incoming information packet should be saved. This decision is made by making use of the information choice mechanism for the task. The information choice mechanism is specified on the task symbol. There are four decision modes which are available for use with the information choice mechanism:

FIR	Retain the information packet accompanying the first satisfied requirement.
LAS	Retain the information packet accompanying the last satisfied requirement.
BIG	Retain the information packet with the biggest value in a given information attribute.
SMA	Retain the information packet with the smallest value in a given information attribute.

If the incoming information packet satisfies the requirements of the decision mode, it is saved. Any previously saved information packet will no longer be saved. If the incoming information packet does not satisfy the decision mode requirements, then it is not saved. When all predecessor requirements have been satisfied, the task will have a single information packet associated with it.

Each task in a SAINT network having multiple predecessor requirements to satisfy should be assigned a decision mode for use by the information choice mechanism. If no decision mode is prescribed by the user on input, decision mode LAS is selected as a default condition.

The representation of the information choice mechanism on the task symbol is presented in Figure 22. The task description code for the information choice mechanism is INCM. In the *right-hand section* of the row for the information choice mechanism, we enter the information choice mode we have selected. In Figure 22, task 1 retains the information packet accompanying the first satisfied requirements; task 2 retains the information packet accompanying the last satisfied requirement; task 3 retains the information packet with the biggest value of information attribute 1; and task 4 retains the information packet with the smallest value in information attribute 2. An example of the significance of the information choice mechanism will be illustrated in a later section of this manual.

Summary

1. Information flows from task to task in a manner dictated by the precedence relations.

2. Information is organized into packets, with each information packet containing attributes.

3. The packets flowing through the network are differentiated by the values of information attributes.

4. At the start of a simulation, information packets are created at all source tasks.

5. The number of attributes contained in each information packet is specified on input.

6. When a task is completed, the information packet associated with that task is transmitted along each precedence branch selected by the branching operation.

7. Assignments of information attribute values are made at the tasks of the network.

8. The task description code for an attribute assignment is ATAS.

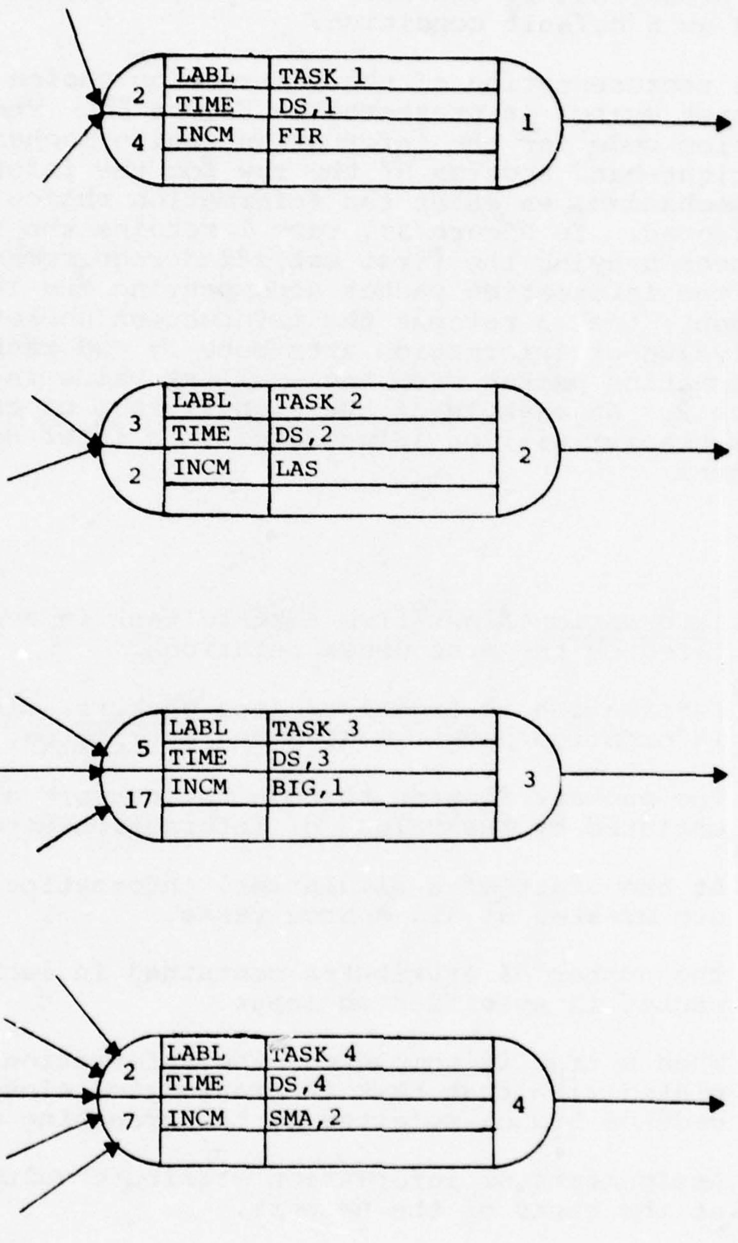


Figure 22. SAINT Model Illustrating Information Choice Mechanism Specifications.

9. The description of the assignment appears on the right-hand side of the ATAS row.

10. Attribute assignments at a task can be made at the release of the task, at the start of the task, or at the completion of the task.

11. To make an information attribute assignment, the specification IA is used, followed by the information attribute number that will be assigned a value.

12. The values of information attributes may be assigned as constants or samples from a distribution set, in the same manner as task duration.

13. A SAINT task can have only one information packet associated with it.

14. The information choice mechanism determines which incoming information packet should be saved for a task with multiple predecessor requirements.

15. The information choice mechanism can take one of four decision modes:

FIR	Retain the packet accompanying the first satisfied requirement.
LAS	Retain the packet accompanying the last satisfied requirement.
BIG	Retain the packet with the biggest value in a given information attribute.
SMA	Retain the packet with the smallest value in a given information attribute.

16. If no decision mode is prescribed by the user SAINT selects LAS as the default condition.

17. The task description code for the information choice mechanism is INCM.

18. In the right-hand section of the INCM row, we enter the decision mode desired.

SECTION V

TASK STATISTICS

SAINT is a simulation language designed to be used for the study of systems that contain resources, equipment, and environmental constraints. In general, the most common SAINT usage is the study of resources performing tasks to achieve a desired objective. For this reason, SAINT automatically provides the user with utilization statistics on all defined resources. However, we may also desire some information on how the tasks in the network are performed. While SAINT does not provide these statistics automatically, it does provide a framework through which an analyst can obtain any type of task statistic desired.

For any task in a SAINT network that is defined as a *statistics task*, SAINT obtains estimates of the mean, standard deviation, minium, maximum, and a histogram associated with the *statistical quantity* to be observed. The specification of the statistical quantity is fairly complex, since SAINT provides a great deal of flexibility in the specification of the statistics to be collected.

Defining a Statistical Quantity

Essentially, the statistical quantity is defined by a *statistic type* and a *collection point*. The types of statistics that can be collected are:

FIR	The time of the first occurrence
ALL	The time of all occurrences
BET	The time between occurrences
NUM	The number of occurrences

After reading the above, the logical question is: What is an occurrence? An occurrence is defined as one of the following:

REL	The release of a task
STA	The start of a task
COM	The completion of a task
CLR	The clearing of a task

(At this point, we have yet to define task clearing. Clearing will be defined later, and it is sufficient to simply note here that statistics can be collected at task clearings.)

If we combine the definitions for statistic type and occurrence time, we can see that the following are possible statistics we can collect on tasks:

FIR REL	The time of the first release of a task
ALL COM	The time of all completions of the task
BET STA	The time between starts of a task
NUM CLR	The number of clearings of a task

In fact, there are sixteen possible combinations of statistics that can be collected at any one task. However, only one of the combinations can be requested for any one task in the network. The following table lists all possible statistical collection combinations.

	RELEASE	START	COMPLETION	CLEARING
FIRST	FIR REL	FIR STA	FIR COM	FIR CLE
ALL	ALL REL	ALL STA	ALL COM	ALL CLE
BETWEEN	BET REL	BET STA	BET COM	BET CLE
NUMBER	NUM REL	NUM STA	NUM COM	NUM CLE

Identifying a Statistics Task

For our GCV model in Figure 21, we would like to collect statistics on the time between initiation of GCV flight (task 3), the time that the launch process for GCV 2 begins (task 2), and the number of times GCV status is recorded (task 4). The task description code for task statistics is STAT. The definition of the statistic to be collected is placed on the right-hand side of the STAT row. Figure 23 displays the GCV model with the statistics collection requirements included. This model will collect and report statistics on the time of the first start (FIR STA) of task 2, the time between releases (BET REL) of task 3, and the number of completions (NUM COM) of task 4.

Interval Statistics

In examining the SAINT statistic collection procedure described above, we note that the available statistics do not represent any interactions or relations between tasks. For example, suppose that we wanted to know how long it took for operator 2 to launch both GCVs. In this case, we would like to know the interval of time between the start

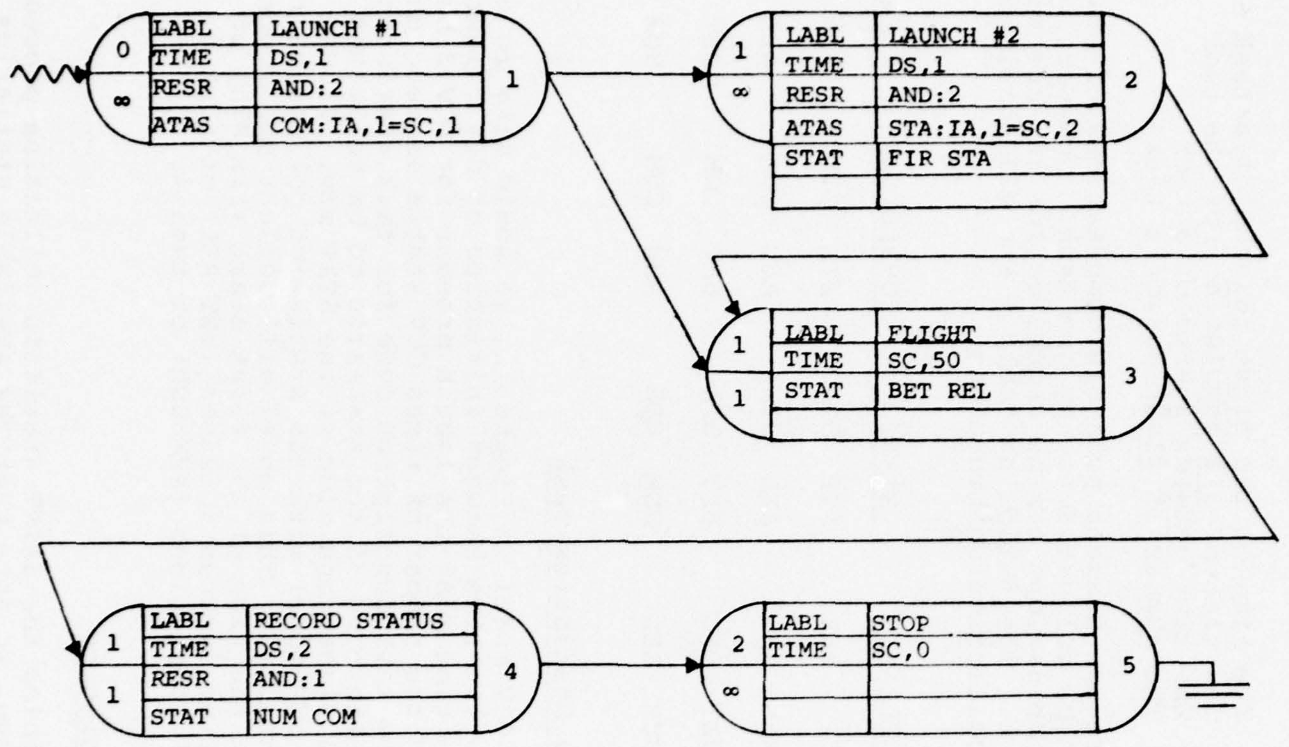


Figure 23. SAINT Model Illustrating Task Statistics.

of task 1 and the completion of task 2. Because this interval of time type of statistic is of interest, SAINT allows us to specify interval (INT) statistics in addition to number, between, first, and all statistics.

As with the other statistics, interval statistics may be collected at the release, start, completion, or clearing of any task in the network. The specification of interval statistics at a task requires a reference time to be subtracted from the current time. The reference time is called the mark time and its value flows through the network with the information packet. The resultant value of the subtraction is the interval of time that is collected.

Assigning Mark Times

Mark times are values of time that are carried with the information packets as they flow through the network. We can, at various points in the network, assign the mark times. The possible assignment points are:

FPC	The first predecessor completion of a task (e.g., if three predecessors were required, time would be marked upon completion of the first predeecessor)
REL	The release of a task
STA	The start of a task
COM	The completion of a task

We can obtain information on the time it takes an information packet to flow from one point in a network to another by specifying a mark at the first point and collecting an interval statistic at the second. For the example where we desire the time to launch both GCVs, we would mark at the start of task 1 and collect interval statistics at the completion of task 2. This is shown in Figure 24. Note that the task description code for marking is MARK and the right-hand side of the row for marking contains the occurrence time at which marking for this task is to take place.

In SAINT, marking and statistics collection may be performed at the same task. Thus, we may mark at the first predecessor completion of a task and collect interval statistics at the same task's completion. We may also mark at the release of the task and collect interval statistics at the start of the task. This causes the collection of a statistic representing the time the task awaits a resource before it is started. With the ability to mark and to collect first, all, between, number, and interval statistics at various points for each task, SAINT provides the capability to abstract a great deal of information from a network simulation.

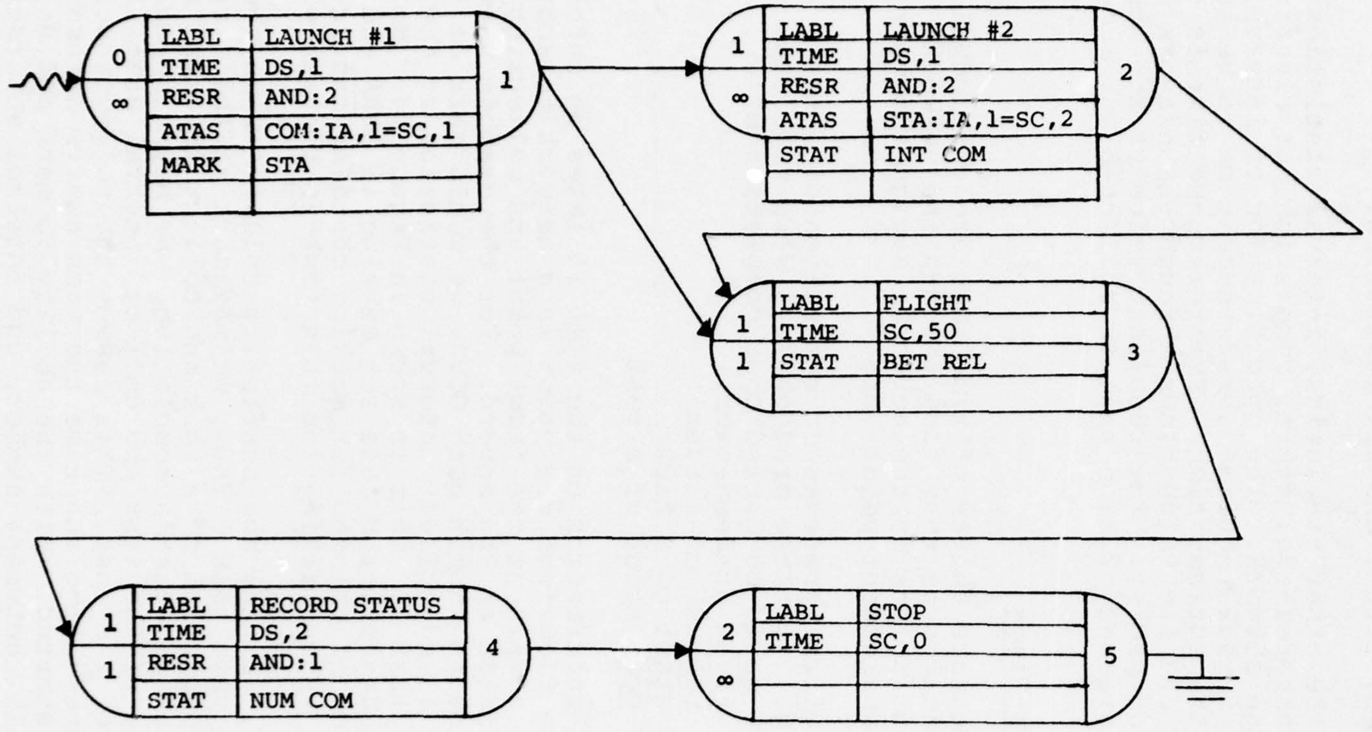


Figure 24. SAINT Model Illustrating Marking and Interval Statistics.

Statistics Task Reports

Statistics task reports can be obtained in both graphical and tabular form. These reports may be obtained for particular iterations or summarized over all iterations. To illustrate these reports, 100 iterations of the SAINT model in Figure 23 were performed.

In the graphical form, a histogram is used to portray the data collected during the simulation. The histogram of the average time between release statistic for task 3 (Flight) over the 100 iterations is shown in Figure 25 Both the relative and cumulative frequency distributions are plotted.

In the tabular form, the mean, standard deviation, and minimum/maximum values observed are used to portray the data. Figure 26 illustrates the statistic task report for the 100 iterations of the SAINT model in Figure 23

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, reread this section.

1. For any task in a SAINT network that is defined as a statistics task, SAINT obtains estimates of the mean, standard deviation, minimum, maximum, and a histogram associated with the statistical quantity to be observed.

2. The statistical quantity is defined by a statistics type and a collection point.

3. The available statistics types are:

FIR	The time of first occurrence
ALL	The time of all occurrences
BET	The time between occurrences
INT	The time interval between the mark time and the collection point
NUM	The number of occurrences

4. The available collection points are:

REL	Release of the task
STA	Start of the task
COM	Completion of the task
CLR	Clearing of the task

```
***HISTOGRAM OF THE AVERAGE BET REL STATISTIC FOR TASK    3 (FLIGHT  )***

OBSV    RELA    CUML         UPPER
FREQ    FREQ    FREQ    CELL LIMIT   0         20        40        60        80        100
                                     +    +    +    +    +    +    +    +    +    +    +
   0       0       0             0   +                                                 +
   0       0       0    1.0000E-01   +                                                 +
   0       0       0    2.0000E-01   +                                                 +
   0       0       0    3.0000E-01   +                                                 +
   0       0       0    4.0000E-01   +                                                 +
  19    .190    .190    5.0000E-01   ++++++++++                                        +
   7    .070    .260    6.0000E-01   ++++         C                                    +
   7    .070    .330    7.0000E-01   ++++            C                                 +
   7    .070    .400    8.0000E-01   ++++                C                             +
   8    .080    .480    9.0000E-01   +++++                   C                         +
   4    .040    .520    1.0000E+00   +++                       C                       +
   2    .020    .540    1.1000E+00   ++                         C                      +
   4    .040    .580    1.2000E+00   +++                          C                    +
  11    .110    .690    1.3000E+00   ++++++                            C               +
  11    .110    .800    1.4000E+00   ++++++                                  C         +
   3    .030    .830    1.5000E+00   ++                                       C        +
   8    .080    .910    1.6000E+00   +++++                                        C    +
   2    .020    .930    1.7000E+00   ++                                            C   +
   5    .050    .980    1.8000E+00   +++                                              C+
   2    .020   1.000    1.9000E+00   ++                                                C
   0       0   1.000    2.0000E+00   +                                                 C
   0       0   1.000           INF   +                                                 C
 ---                                 +    +    +    +    +    +    +    +    +    +    +
 100                                 0         20        40        60        80        100
```

Figure 25. Histogram for the Statistic Collected at Task 3 in the SAINT Model in Figure 23.

```
                           ***STATISTICS TASK SUMMARY REPORT***

                     *AVERAGES OF THE STATISTICS COLLECTED FOR  100 ITERATICNS*

 TASK     TASK     STAT  COLCT     --------STATISTICS ON THE AVERAGE VALUE PER ITERATION---------
NUMBER    LABEL    TYPE  POINT      AVERAGE      STD DEV     NO. ITER     MINIMUM      MAXIMUM

   2     LAUNCH2    FIR   STA     1.0237E+00   3.9329E-01     100      5.0000E-01   2.0000E+00
   3     FLIGHT     BET   REL     1.0190E+00   4.2657E-01     100      5.0000E-01   1.8809E+00
   4     STATUS     NUM   COM     2.0000E+00            0     100      2.0000E+00   2.0000E+00
```

Figure 26. Statistics Task Summary Report for the SAINT Model in Figure 23.

5. The task description code for task statistics is STAT.

6. In the right-hand side of the STAT row, the statistic desired is defined.

7. The mark time associated with interval (INT) statistics is associated with the information packet.

8. Mark times can be assigned at any one of four points in time relating to a task:

FPC	The first predecessor completion
REL	The release of the task
STA	The start of the task
COM	The completion of the task

9. The combination of mark time and interval statistics allows the collection of statistics on the time required for an information packet to flow from one point in a network to another.

10. The task description code for marking time is MARK.

11. The right-hand side of the MARK row contains the time at which the marking is to take place.

12. Marking and statistics collection may be performed at the same task.

SECTION VI

TASK PRIORITY

Suppose that instead of requiring operator 2 to launch the first GCV and then launch the second GCV, as dictated by the precedence relations of Figure 24, we don't specify a task sequence but only require him to launch both GCV 1 and GCV 2. In this case, both task 1 and task 2 will be source tasks that have branching from them to task 3. This situation is illustrated in Figure 27. However, suppose that we still want to retain control over which task is performed first. In some cases, we would like task 1 to be performed before task 2. In other cases, perhaps we would like task 2 to be performed first. Thus, we would like some way of informing resource 2 that one task is more important than another task. This is accomplished through the use of task priority.

Task Scheduling

Each task in a SAINT network can be assigned a priority. When two tasks require the same resource at the same time, the resource will decide which task to perform on the basis of priority. In general, the task with the highest priority will be performed first. In our situation, if we assign a priority of 1.0 to task 1 and a priority of 0.5 to task 2, operator 2 will launch the first GCV prior to launching the second. On the other hand, if the priorities are reversed, then the operator will launch the second GCV first. The use of priorities in the SAINT model allows us flexibility in determining the way the system operates.

Assignment of Task Priority Values

The priority specification for a task, as given in Figure 27 , is presented in the task description portion of the task symbol. (More complex task priority concepts will be discussed in Section XIV.) The task description code for task priority is PRTY. The priority we assign the task is then given in the right-hand side of the row defining task priority. In Figure 27, we have defined the priority of task 1 to be 0.5 and the priority of task 2 to be 1.0. Thus, for the case presented, GCV 2 will be launched first and GCV 1 will be launched second (note that we no longer collect an interval statistic at task 2, since this task is no longer a predecessor task to task 1).

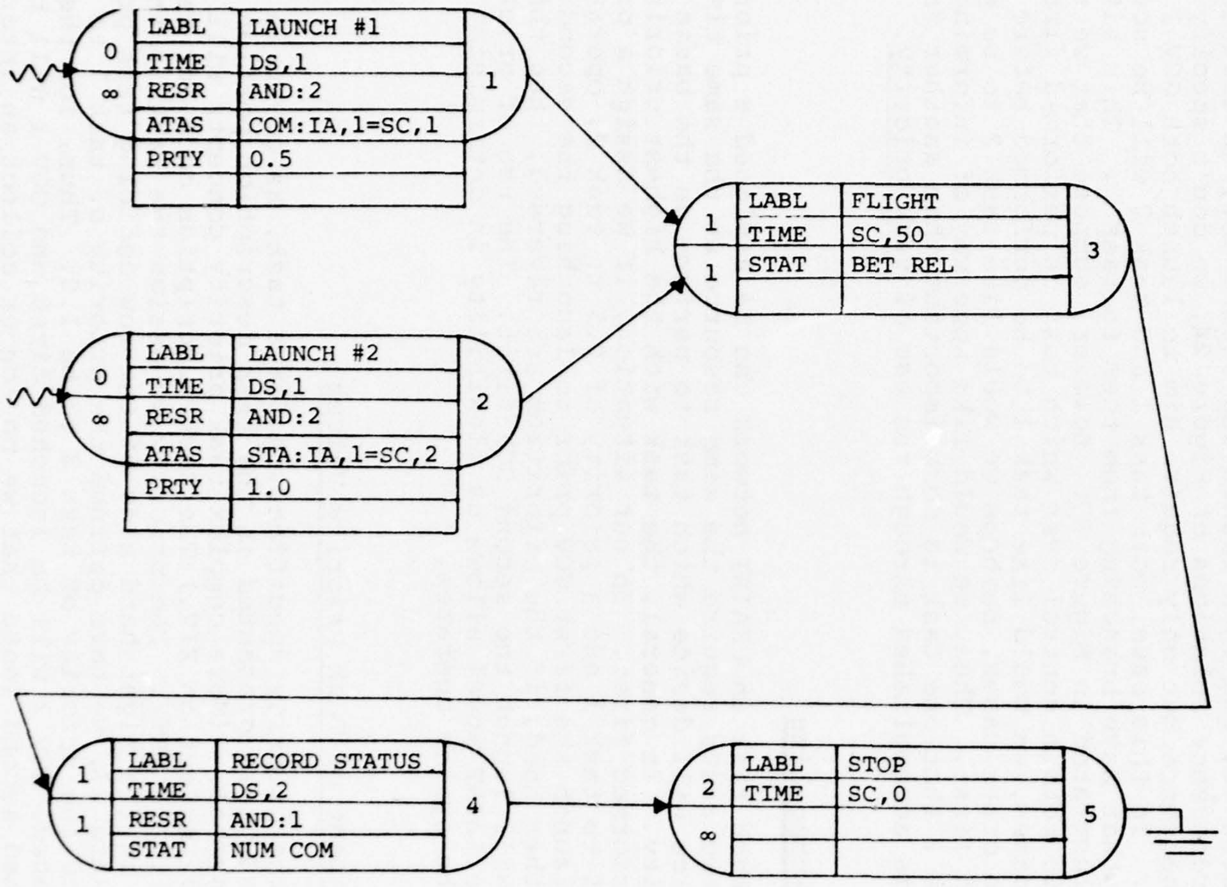


Figure 27. SAINT Model Illustrating Task Priority.

Task Priority and the Information Choice Mechanism

Now, in addition to the statistics we are already collecting, suppose that we are interested in determining the interval of time between the time of launch of the last GCV to the completion of the mission. We might say, "Simple enough, we simply mark at the completion of task 1 and collect interval statistics at the completion of task 5." True, but we must now be concerned with the information packet that is retained by task 5. Remember that two predecessor completions are required for the release of task 5. The second GCV will cause the first predecessor completion of task 5 while the first GCV will cause the second predecessor completion of task 5. Since we are marking at task 1, we would like to retain the information packet associated with task 1, that is, the information packet associated with GCV 1.

In this case, we have two options of specifying the information packet to be kept by task 5. First, since we know that the information packet that we desire (that of GCV 1) will cause the second release of task 5, we can simply use the LAS mode for the information choice mechanism. Second, since we know we want to retain the packet of GCV 1, we can specify that we would like to retain the packet with the smallest number in the information attribute 1 (SMA,1). Using the latter specification, the GCV model that results appears in Figure 28. Naturally, if we change the priorities of tasks 1 and 2, we must also be concerned with a change in the information choice mechanism at task 5 to insure that the statistic produced by SAINT is the statistic that we desire.

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, re-read the section.

1. Each task in a SAINT network can be assigned a task priority.

2. When two or more tasks require the same resource at the same time, the resource will perform the task with the highest priority first.

3. The task description code for the task priority is PRTY.

4. The right-hand side of the PRTY row contains the numeric value of the task priority.

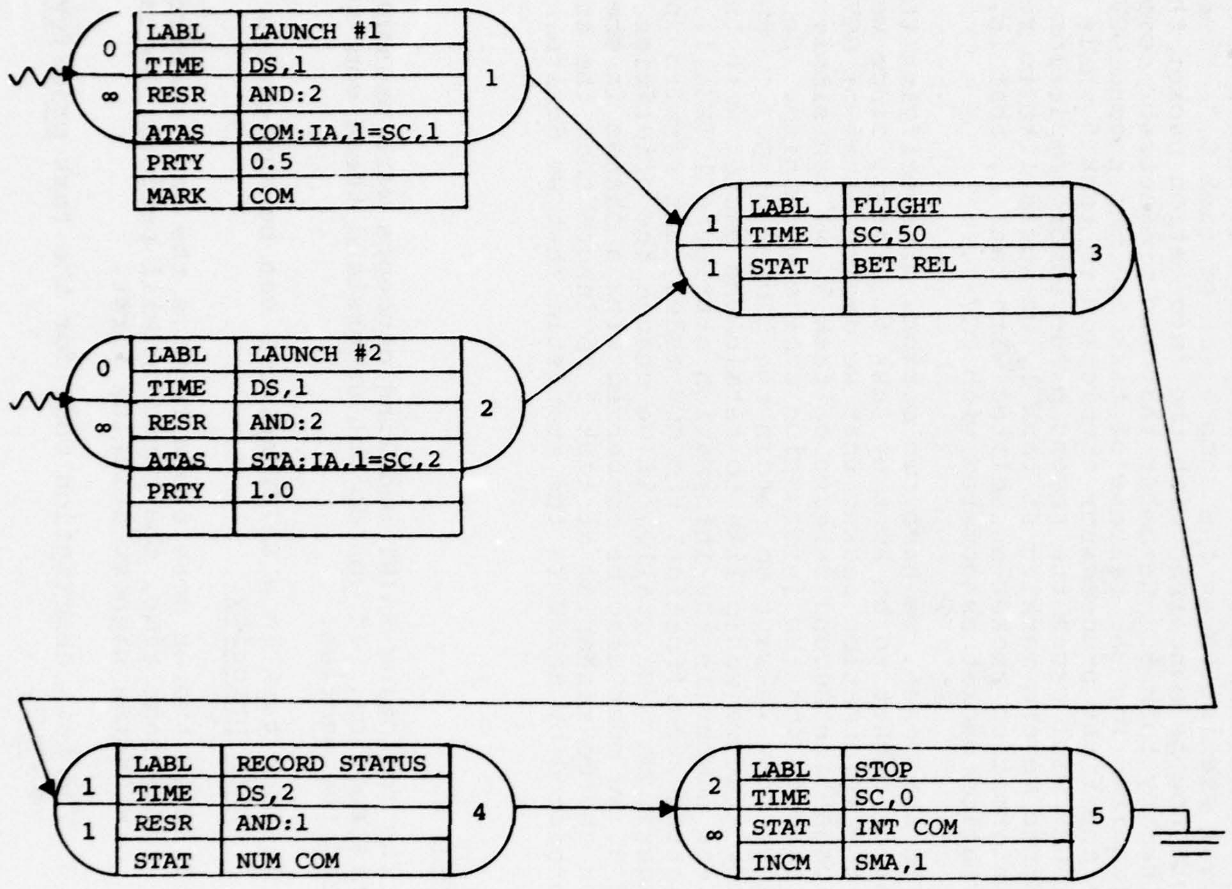


Figure 28. SAINT Model Illustrating the Use of the Information Choice Mechanism.

SECTION VII

RESOURCE ATTRIBUTES

Up to this point, we have included various types of resources in our SAINT models, such as human operators and equipment (e.g., computers). The tasks for which resources were specified could only be performed if the required resources were available, that is, they were not working on other tasks. However, we had no means for differentiating among the resources other than by resource number, although they can be uniquely described by a set of characteristics. Human operators can be described by height, weight, level of training, etc. Characteristics of computer equipment include processing speed and available storage capacity. Thus, the characteristics of a resource can affect task performance. In SAINT, resource attributes, designated RA, are the means by which we assign characteristics to resources. Only those characteristics which affect task performance need be included as resource attributes in a model.

Description of Resource Attributes

Each resource included in a SAINT model is described by a set of attributes associated with it. These attributes are organized into packets, with each packet characterizing a particular resource. At the start of a simulation, resource packets are created for each resource included in the model. Each packet contains attributes which uniquely describe the associated resource. Each packet remains with its associated resource throughout the simulation.

Depending on the type of resource, resource attributes can represent many different characteristics. If the resource is a hammer, we may assign resource attributes describing its length, its weight, whether or not it has a claw, or perhaps what material it is made of. On the other hand, if the resource is a computer, it may be described by its processing speed or the amount of available memory. In many cases, as in the GCV system, the resource is a human operator. For an operator, we might include his weight, his height, or his level of intelligence.

Resource Attributes for the GCV System

In the real world, some operators are faster or slower than others. For illustrative purposes, let us assume that we can assign a numeric value to the relative speed of operators. This "speed factor" defines the speed of an operator's performance in relation to a "normal" operator. Thus, if an operator has a speed factor of 0.9 and the "normal" operator performs a task in ten minutes, then our operator will perform the task in nine minutes. Similarly, if his speed factor were 1.1, he would perform the task in eleven minutes.

In order to demonstrate the use of resource attributes in a SAINT model, assume that only one operator is required in our GCV model, and that he performs the launch and monitoring operations for both GCVs. Our objective is to simulate this system using operators with varying speed characteristics in order to determine the effect of this characteristic on the time that it takes to complete the entire mission. Thus, we will collect interval statistics from the start of the simulation until the end of the simulation. By comparing the statistics generated based on operators having different speed factors, we can determine the effect of operator speed on mission performance.

Assignment of Resource Attribute Values

In order to perform this experiment, we must first define speed as an attribute of the resource. How do we do this? The attribute assignment mechanism allows us to selectively assign or change the values of resource attributes at any task included in our model. In addition, assignments to individual attributes of any resource included in the model can be made prior to the start of the simulation. For a resource attribute assignment at a task, it is not necessary for the resource to be working on that task in order for assignments to be made to one or more of its attributes.

As with information attributes, we have three options in making a resource attribute assignment at a particular task in the network. Assignments can be made at the release (REL) of a task, at the start (STA) of a task, or at the completion (COM) of a task. However, if more than one type of attribute assignment is made (information or resource) at a particular task, the point of assignment option applies to all attribute assignments. In other words, all attribute assignments made at a task take place at the same point in relation to task performance.

For a resource attribute assignment, as with the information attribute assignment, the ATAS description code is used. Four pieces of information are required to specify a resource attribute assignment. These are a resource number, an attribute number, a function type, and a parameter specification. The resource number, which follows the RA specification in the right-hand row of the assignment, defines the resource for which the assignment is to be made. The resource attribute number, which follows the resource number, indicates the attribute of the specified resource's attribute packet to which the assignment is to be made. For example, RA 4-17 would be used in specifying assignment to attribute 17 of resource 4. Like information attribute assignments, the function type is used in conjunction with the parameter specification in order to generate the value to be assigned to the attribute. The parameter specification may either indicate the distribution set which contains the parameters for the assignment, or it may be used as a parameter in the assignment function. The function type and parameter specification are the same concepts used when specifying task duration. Thus, the SAINT user can perform an assignment using any of the available SAINT distributions or functions.

SAINT Model of the GCV System

Figure 29 presents the SAINT model for our newly hypothesized GCV system. The single resource included in the model is required to perform tasks 1, 2, and 4. The statistic of interest is the time interval from the start to the completion of the mission. Thus, we mark at the release of task 2 (task 2 will be the first task performed due to its higher priority) and we collect an interval statistic upon the completion of task 5. Since we want to use the mark time assigned at task 2 in our statistics calculations, we employ the information choice decision mode BIG,1. Finally, we assign a value of 0.9 to attribute 1 of resource 1 at the release of task 2 (distribution set 3 is defined as a constant distribution with a value of 0.9). In our model, we have now defined resource attribute 1 as the speed factor for resource 1.

We have now built a model that can be used to analyze the effect on mission performance of operators with different speed factors. However, in order to continue with the experiment, we must specify the effect of this speed factor on task performance. The method for doing so is the subject of the next section.

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, re-read the section.

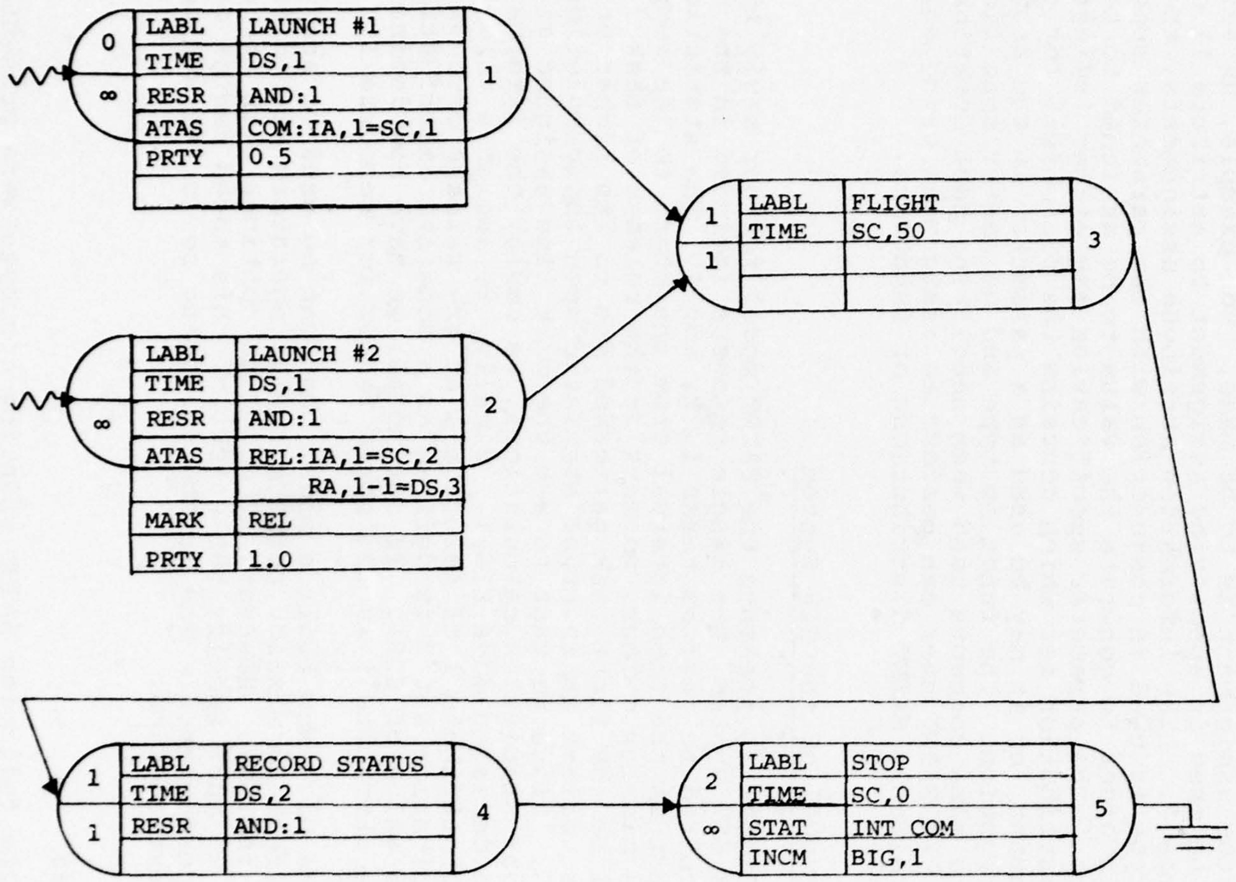


Figure 29. SAINT Model Illustrating a Resource Attribute Assignment.

1. Resource attributes are the means by which characteristics are assigned to resources.

2. Resource attributes are organized into packets, with each packet characterizing a particular resource.

3. At the start of a simulation, resource packets are created for each resource included in the model.

4. Each packet remains with its associated resource throughout the simulation.

5. The attribute assignment mechanism is used to selectively assign or change resource attribute values at any task in the network.

6. Assignments to resource attributes can be made prior to the start of a simulation.

7. It is not necessary for a resource to be working on a task in order for values to be assigned to its attributes at that task.

8. Resource attribute assignments can be made at the release (REL), start (STA), or completion (COM) of a task.

9. All attribute assignments made at a task, regardless of type, must be made at the same point in relation to task performance.

10. The task description code for an attribute assignment is ATAS.

11. To make a resource attribute assignment, the specification RA is used, followed by the resource number and resource attribute number.

12. Values to be assigned to resource attributes are determined by a function type and parameter specification in the same manner as information attributes.

SECTION VIII

MODERATOR FUNCTIONS

In our previous GCV models, the time required to perform a task was either a constant or a sample drawn from a probability distribution. However, as we have seen in the previous section, task duration is not always solely dictated by the value of the constant or the sample drawn from the designated probability distribution. Thus, in order to fully represent complex man-machine systems, it is often necessary to modify task duration, as well as other task performance characteristics, as a function of resource characteristics and/or system status. In SAINT, any function that specifies an effect on baseline task performance is defined as a moderator function.

Incorporating Moderator Functions in a SAINT Model

Subroutine MODRF is provided by SAINT in order for us to define any number of moderator functions. If we wish to incorporate moderator functions in our model, we must code the moderator function logic in subroutine MODRF in FORTRAN or a compatible language. All standard language conventions must be observed. The form to be used in writing subroutine MODRF is shown in Figure 30.

Subroutine MODRF is called automatically by SAINT for the tasks that we specify at the time that those tasks are being scheduled for performance (after resources have been assigned). The subroutine has two arguments: MODFN is the number of the moderator function to be applied; and NTASK is the task number for which the moderator function is being called. A computed GO TO statement can be used in order to transfer control within subroutine MODRF to the appropriate moderator function code. If more than one moderator function is to be applied to the task, SAINT calls subroutine MODRF the required number of times with the appropriate arguments.

Referring to Figure 30, assume that moderator functions 1 and 3 are to be applied to a hypothetical task 21. At the time of scheduling task 21, SAINT executes the statement CALL MODRF(1,21). Control is then transferred to subroutine MODRF, the FORTRAN code beginning with statement 10 is executed, and control is returned to SAINT. SAINT then executes the statement CALL MODRF(3,21). Control is once again transferred to subroutine MODRF, the FORTRAN code beginning with the statement 30 is executed, and control is returned to SAINT. Task 21 is then scheduled for performance.

```
      SUBROUTINE MODRF(MODFN,NTASK)

      *************
      COMMON CARDS  (if necessary)
      *************

      GO TO (10,20,30),MODFN

   10 ** FORTRAN code for moderator function 1
      RETURN

   20 ** FORTRAN code for moderator function 2
      RETURN

   30 ** FORTRAN code for moderator function 3
      RETURN
      END
```

Figure 30. General Form of Subroutine MODRF, Illustrating Three Moderator Functions.

We may require information other than the moderator function number and the task number in order to fully define our moderator functions. For this purpose, we may include SAINT labeled COMMON storage in subroutine MODRF. Two variables of importance that are located in SAINT COMMON storage are TTIME and PFIRB. When subroutine MODRF is called, TTIME contains the baseline task duration (the constant or sample from a probability distribution) and PFIRB contains the probability of selecting the first branch specified upon completion of task NTASK. Of course, the use of PFIRB is only applicable when task NTASK required probabilistic branching. Otherwise, PFIRB will be assigned a value of 1.0.

Moderator Function Status Specification

Each moderator function included in a SAINT model is identified by a unique moderator function number. The total number of moderator functions, as well as the initial status of each moderator function, is specified on input. Moderator functions can initially be either active or inactive. If a moderator function is initially active, it will be applied to every task in the network until its status has been changed. If it is initially inactive, it will not be applied to any task until a change to active status is specified. Unless we specify otherwise, all moderator functions are assumed to be initially inactive.

Changes to the status of any moderator function included in a model can be made at any task in the network. These changes can be permanent (until another status change is made or the present iteration ends) or only for the task at which the specification is made (following scheduling of the task, the status of the moderator function returns to its previous state). Status specifications made at a task, if any are made, will override initial or previous specifications if a conflict arises. The SAINT User's Manual describes the procedure for specifying moderator function status.

Moderator Functions for the GCV System

For the GCV system described in the last section, we require a moderator function that specifies the effect of the speed factor (resource attribute 1) on the baseline task performance times of tasks 1, 2, and 4, since each of those tasks require resource 1. The network shown in Figure 31 illustrates the means by which we specify the moderator functions applicable to each task. The task description code

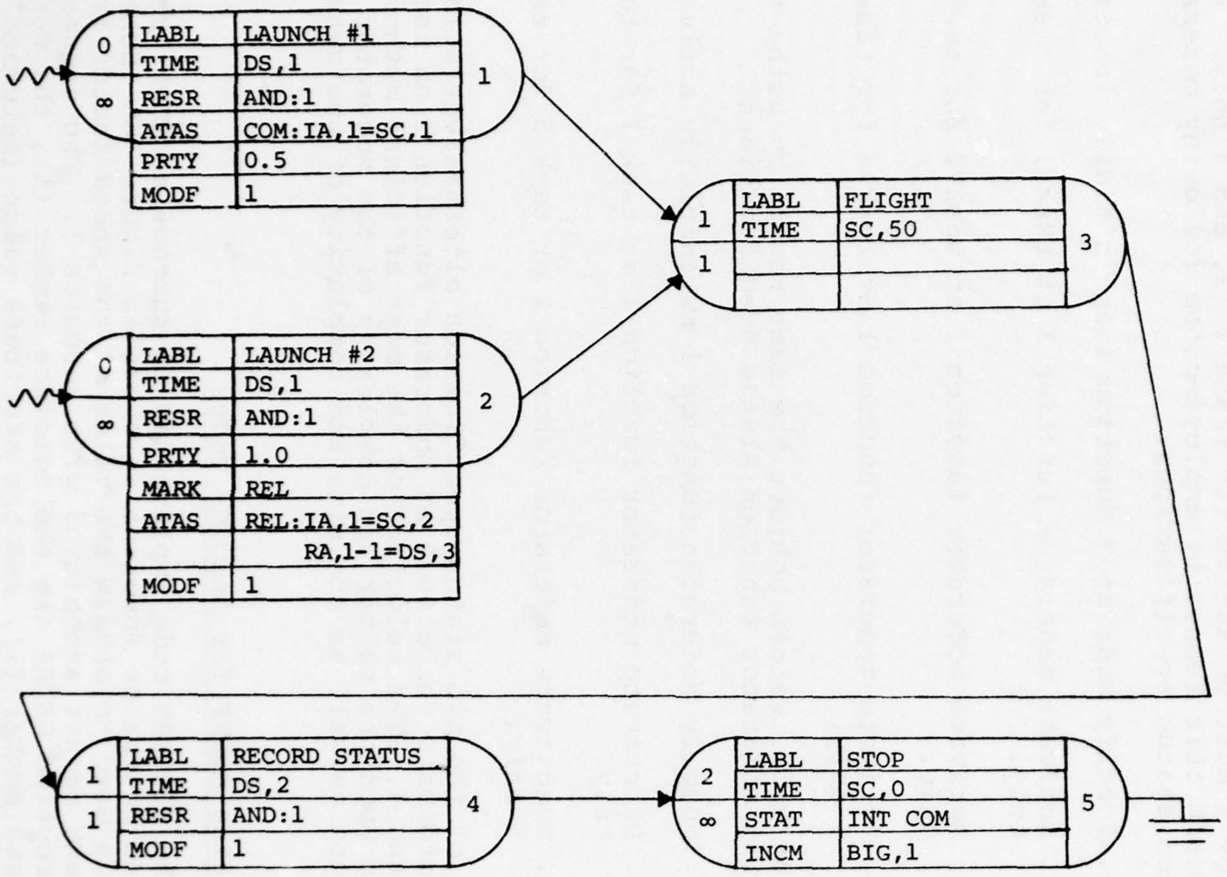


Figure 31. SAINT Model Illustrating Moderator Function Specification.

for moderator functions that apply to a task is MODF. The number(s) of the moderator function(s) that apply to the task are then listed on the right hand side of the MODF row. For our GCV example in Figure 31, we see that moderator function 1 is to be applied to tasks 1, 2, and 4.

The requirements of our model dictate that moderator function 1 must be active for tasks 1, 2, and 4 only. We can obtain this result by employing the following moderator function status specifications:

1. Specify moderator function 1 as initially inactive.

2. Activate moderator function 1 at task 1 for task 1 only.

3. Activate moderator function 1 at task 2 for task 2 only.

4. Activate moderator function 1 at task 4 for task 4 only.

Alternatively, we can achieve the same result by using the following moderator function status specifications:

1. Specify moderator function 1 as initially active.

2. Deactivate moderator function 1 at task 3 for task 3 only.

3. Deactivate moderator function 1 at task 5 for task 5 only.

Either of the two status specification alternatives allow us to incorporate the effect of moderator function 1 on tasks 1, 2, and 4. The selection of the most efficient method depends upon the number and complexity of the moderator functions as well as the size and complexity of the network model.

Subroutine MODRF for the GCV Model

The FORTRAN code implemented in subroutine MODRF for the required moderator function is shown in Figure 32. Subroutine GETRA is used to obtain the value of the speed factor from the attribute packet associated with resource 1. The arguments to subroutine GETRA are the resource number (1), the resource attribute number (1), and the attribute value (returned by subroutine GETRA). Then, to obtain the desired moderator

```
      SUBROUTINE MODRF(MODFN,NTASK)
C
      COMMON /COM22/ TTIME,PFIRB
C
C**** MODIFY TASK PERFORMANCE TIME BY SPEED FACTOR
C
      CALL GETRA(1,1,VALUE)
      TTIME=TTIME*VALUE
      RETURN
      END
```

Figure 32. Subroutine MODRF for GCV System of Figure 25.

function effect, we simply multiply TTIME (the baseline task performance time) by VALUE (the value of the speed factor). The new value of TTIME is used by SAINT as the time required to perform the task. You will note that unlike the design presented in Figure 30, the argument MODFN is not used in our moderator function, since we have only one moderator function included in the model.

Summary

The following SAINT modeling concepts were presented in this section. If you don't understand these concepts, re-read the section.

1. Any function that specifies an effect on a baseline task performance is defined as a moderator function.

2. All required moderator function logic is coded in FORTRAN or a compatible language in subroutine MODRF.

3. Subroutine MODRF is automatically called by SAINT for the task specified with the moderator function number and task number as arguments.

4 . A computed GO TO statement is used to transfer control within subroutine MODRF to the appropriate moderator function code.

5. If more than one moderator function is to be applied to a task, SAINT calls surbroutine MODRF the required number of times with the appropriate arguments.

6. The SAINT COMMON variable TTIME is assigned the value of baseline task duration prior to the call to subroutine MODRF.

7. The SAINT COMMON variable PFIRB is assigned the probability of selecting the first branch emanating from a task that requires probabilistic branching. If a task does not require probabilistic branching, PFIRB will be assigned a value of 1.0.

8. Each moderator function included in a SAINT model is identified by a unique moderator function number.

9. The total number of moderator functions, as well as the initial status of each moderator function, is specified on input.

10. Moderator functions can initially be either active or inactive.

11. Changes to the status of any moderator function included in a model can be made at any task in the network.

12. Changes to the status of moderator functions at a task can be permanent or only for the task at which the specification is made.

13. Status specifications made at a task will override any previous specifications if a conflict arises.

14. The task description code for moderator functions that apply to a task is MODF. The number(s) of the moderator function(s) that apply to the task are listed on the right-hand side of the MODF row.

SECTION IX

SYSTEM ATTRIBUTES

In previous sections, we have modeled the GCV system in which one operator launches two GCVs and then records their flight deviation after they have flown for at least 50 time units. We have incorporated the capability to model different operators performing these tasks through the use of a moderator function and resource attributes. In addition, our model records statistics on the length of time it takes the operator to complete his required tasks. Although we have included the characteristics of the operator in our model, we have yet to include the characteristics of the GCV.

In many situations, it may be desirable to specify attributes which are not directly applicable to an information-oriented or resource-oriented characterization. These attributes, being global in nature, do not flow through or move about the network as information and resource packets do. These attributes are characterized as system attributes, with one set of system attributes being associated with the SAINT model.

System Attributes for the GCV System

For our GCV system, we define four system attributes. The velocity of GCV 1, 500 feet per second, is stored in system attribute 1. The velocity of GCV 2, 500 feet per second, is stored in system attribute 2. The headings of the two GCVs are stored in system attributes 3 and 4, respectively. GCV heading is defined in terms of radians with respect to a due east reference (a heading of $\pi/2$ radians is a due north heading, while a heading of 0 radians is a due east heading). For this example, assume that both GCVs are heading due east.

Assignment of System Attribute Values

The number of system attributes included in a model, as with information and resource attributes, is specified on input. For this example, we specify four system attributes. As with resource attributes, we have the capability to make initial system attribute assignments if desired. We also have the option of making system attribute assignments at any task in the network. Like other attributes, system attribute assignments can be made at either the release, start, or completion of any task. However, remember that all attribute assignments of any type made at a particular task are made at the same point in relation to task performance.

To make a system attribute assignment, the ATAS task description code is used with the RES, STA, or COM assignment point specification. The alphanumeric symbol SA is used to identify a system attribute assignment. The assignment specification includes the system attribute number to be assigned a value, as well as the function type and parameter specification to be used in determining the assignment value.

The system attribute assignments described above are illustrated in Figure 33. System attributes 1 and 3, representing the velocity and heading of GCV 1, are assigned values at task 1. Likewise, system attributes 2 and 4 are assigned values at task 2.

Information, Resource, and System Attributes

With the presentation of system attributes in this section, we now have access to three distinct classes of attributes: information, resource, and system. While each is related to a network model in a different manner, each has a complete set of parallel assignment and usage capabilities.

We have now defined the characteristics of the resource in our model, as well as the characteristics of the GCVs. However, we have yet to see how the characteristics of the GCVs are used to model the flight of the GCVs now represented by task 3. The model of GCV flight is discussed in the next section.

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, reread the section.

1. Attributes which are not directly applicable to an information-oriented (flow through the network) or resource-oriented (move about the network) characterization are called system attributes.

2. A single set of system attributes is associated with the SAINT model.

3. The number of system attributes to be included in the model is specified by the user on input.

4. As with resource attributes, initial system attribute assignments may be made.

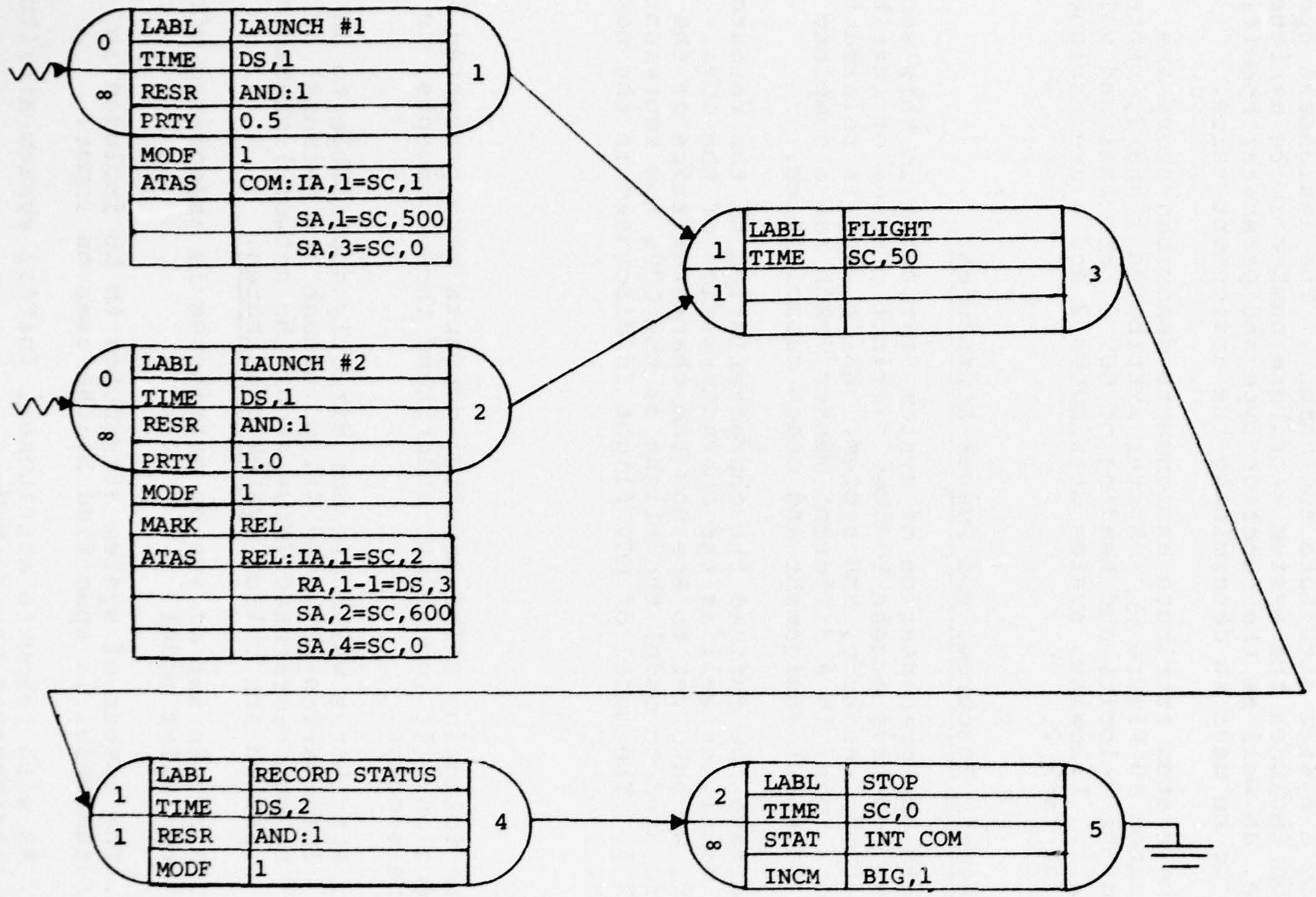


Figure 33. SAINT Model Illustrating System Attribute Assignments.

5. System attribute assignments can be made at the release, start, or completion of any task in the network.

6. The ATAS description code is used with the RES, STA, or COM assignment point specifications to designate a system attribute assignment in the same manner as information and resource attribute assignments.

7. The alphanumeric symbol SA is used to identify a system attribute assignment.

8. The parameters of a system attribute assignment include the system attribute number, and the function type and parameter specification to be used in generating the assignment value.

9. Three distinct classes of attributes (information, resource, and system) are included in SAINT. While each is related to a network model in a different manner, each has a complete set of parallel assignment and usage capabilities.

SECTION X

STATE VARIABLES

When using SAINT, we have the capability to model variables whose values change continuously over time. These variables, called state variables, are descirbed by algebraic, differential, or difference equations that govern their time-dependent behavior. With the incorporation of methods for modeling state variables, SAINT offers those capabilities most often associated with analog or continuous simulation. Moreover, SAINT also provides the constructs for modeling interactions between state variable and task-oriented model components. These concepts will be the subject of this and following sections.

GCV Flight

Let us assume that we are modeling the flight of GCVs as a linear path from their launch point to their final destination, as depicted in Figure 34. The initial position, in x and y coordinates, of the GCV is (x_s, y_s). The final position of the GCV is (x_t, y_t). Thus, the x-position of the GCV at time t(x_t) is equal to x , *the x-position of the GCV at* time s, plus the distance that the GCV flew in the x direction in the time interval between s and t. We now rely on basic trigonometric functions to model the GCV's position over time, given its speed and heading. If h represents the heading of the GCV relative to the x-axis, then x_t will be equal to x_s plus the cosine of the heading h times the velocity v of the GCV times the time interval (t-s). This relation is described by the following equation:

$$x_t = x_s + (\cos (h) * v * (t-s))$$

Similarly, the equation for the y-position of the GCV at time t is:

$$y_t = y_s + (\sin (h) * v * (t-s))$$

State Variable Modeling Concepts

Before presenting the SAINT representation of the GCV flight equations described above, we must define a number of SAINT state variable modeling concepts. First, we have said previously that state variables are defined by writing the difference or derivative equations that describe their time-dependent behavior. These equations must be coded by the user

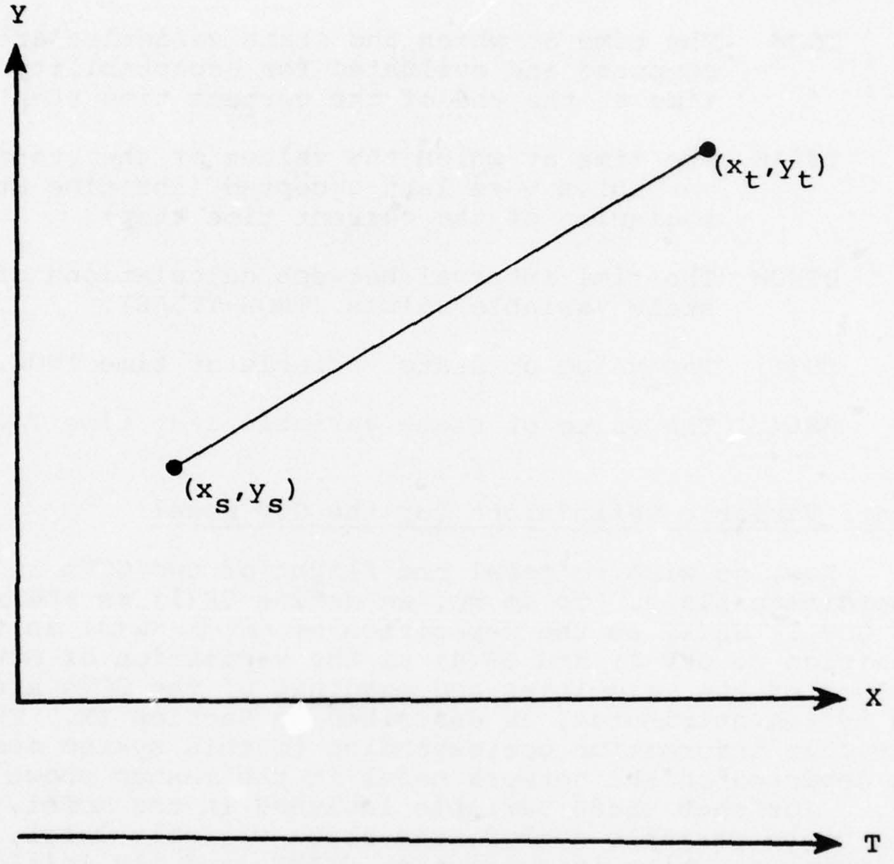


Figure 34. Pictorial Representation of GCV Flight.

in subroutine STATE. This subroutine must be written in FORTRAN or a compatible language. Second, there are a number of SAINT COMMON variables which are necessary in writing the equations that describe our state variables. These variables, included in SAINT labeled COMMON storage, are defined as follows:

TNOW	The time at which the state variables are being computed and evaluated for acceptability (the time at the end of the current time step).
TTLAS	The time at which the values of the state variables were last accepted (the time at the beginning of the current time step).
DTNOW	The time interval between calculations of state variable values (TNOW-TTLAS).
SS(I)	The value of state variable at time TNOW.
SSL(I)	The value of state variable I at time TTLAS.

State Variable Definitions for the GCV Model

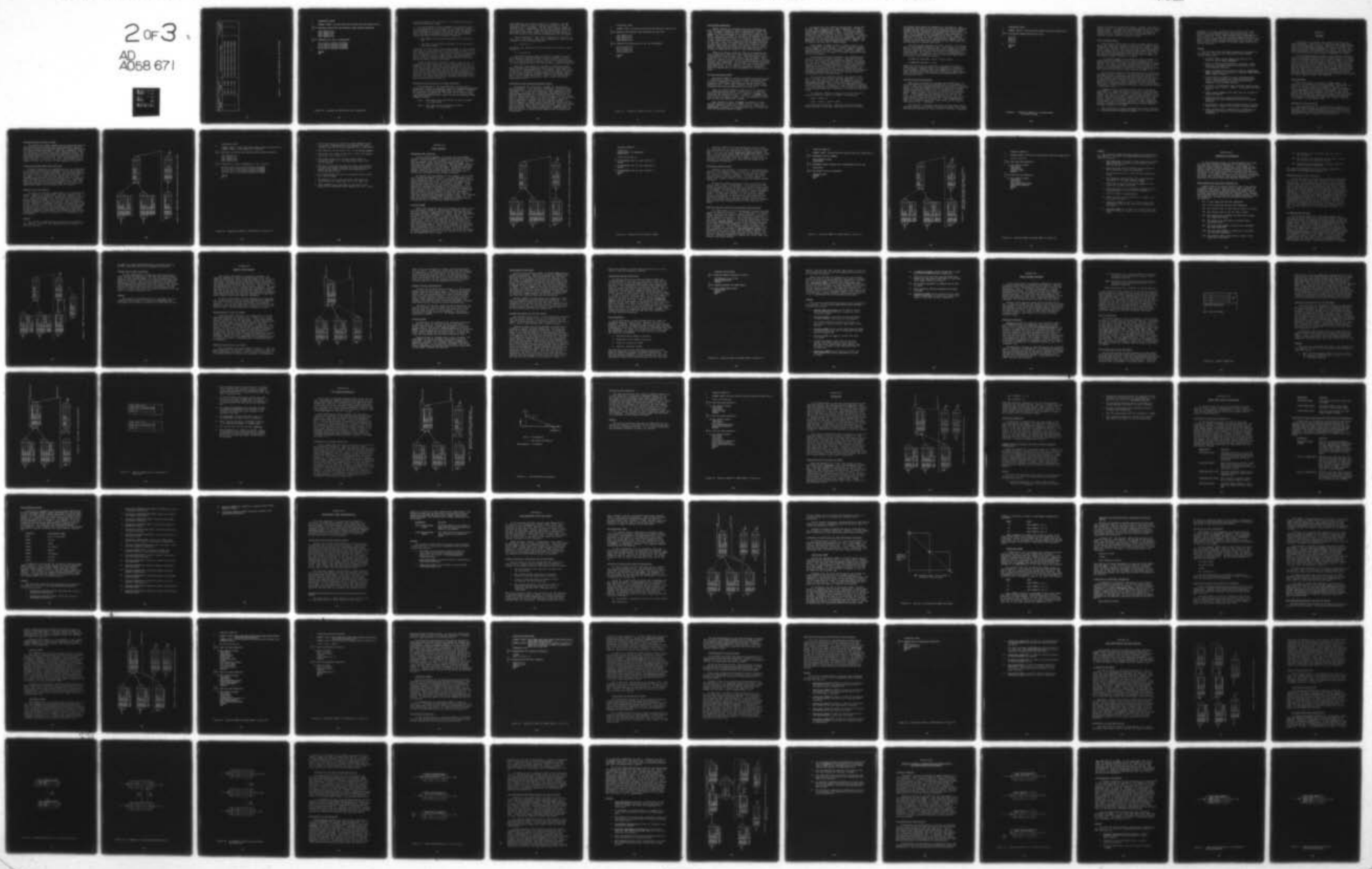
Now, we wish to model the flight of two GCVs in the x-y coordinate plane. To do so, we define SS(1) as the x-position of GCV 1; SS(2) as the x-position of GCV 2; SS(3) as the y-position of GCV 1; and SS(4) as the y-position of GCV 2. The values of the velocities and headings of the GCVs are contained in system attributes, as described in Section IX. The state variable information corresponding to this system description is depicted on the network model in the manner shown in Figure 35. For each state variable included in the model, we give the state variable number, the state variable label, its defining equation in subroutine STATE, and its initial value (to be discussed at the end of this section). You will note that we have abbreviated the phrase "system attribute I" by "SA(I)".

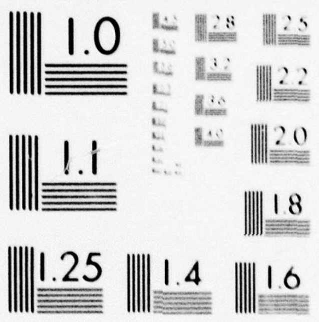
Subroutine STATE for the GCV Model

In order to include state variables in a SAINT model, we must write subroutine STATE. Subroutine STATE for the state variable model component described by Figure 35 is shown in Figure 36. Since the SAINT variables SS(·) and SSL(·) are included in subroutine STATE, we are also required to include SAINT labeled COMMON block COM17. By doing

AD-A058 671 PRITSKER AND ASSOCIATES INC WEST LAFAYETTE IND F/G 9/2
SIMULATION USING SAINT: A USER-ORIENTED INSTRUCTION MANUAL.(U)
JUL 78 D B WORTMAN, S D DUKET, D J SEIFERT F33615-76-C-5012
UNCLASSIFIED AMRL-TR-77-61 NL

2 OF 3
AD
A058 671




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

State Variable Number	State Variable Label	Defining Equation in STATE	Initial Value in INTLC
1	GCV 1: X-POS	SS(1)=SSL(1)+COS(SA(3))*SA(1)*DTNOW	3.
2	GCV 2: X-POS	SS(2)=SSL(2)+COS(SA(4))*SA(2)*DTNOW	7.
3	GCV 1: Y-POS	SS(3)=SSL(3)+SIN(SA(3))*SA(1)*DTNOW	4.
4	GCV 2: Y-POS	SS(4)=SSL(4)+SIN(SA(4))*SA(2)*DTNOW	2.

Figure 35. State Variable Symbolism for the GCV System.

```
      SUBROUTINE STATE
C
      COMMON /COM17/ SS(100),SSL(100),DD(100),DDL(100),LLSVR(100,2)
C
C**** RETRIEVE VELOCITIES AND HEADINGS FROM SYSTEM ATTRIBUTES
C
      CALL GETSA(1,V1)
      CALL GETSA(2,V2)
      CALL GETSA(3,H1)
      CALL GETSA(4,H2)
C
C**** COMPUTE GCV X AND Y COORDINATES
C
      SS(1)=SSL(1)+COS(H1)*V1*DTNOW
      SS(2)=SSL(2)+COS(H2)*V2*DTNOW
      SS(3)=SSL(3)+SIN(H1)*V1*DTNOW
      SS(4)=SSL(4)+SIN(H2)*V2*DTNOW
C
      RETURN
      END
```

Figure 36. Subroutine STATE Using SS(·) Equations.

so, the values of SS(•) and SSL(•) are communicated between SAINT and subroutine STATE.

As stated above, we have assigned the values of the velocities and headings of the two GCVs to system attributes. It is necessary to retrieve these values from the appropriate system attributes in order to define our state variables. This retrieval is accomplished by using the user callable subprogram GETSA, with the following arguments:

1. The number of the system attribute to be retrieved; and

2. The value of the system attribute to be returned by subroutine GETSA.

Following the four calls to subroutine GETSA, V1 will contain the value of system attribute 1 (the velocity of GCV 1), V2 will contain the value of the velocity of GCV 2, H1 will contain the heading of GCV 1, and H2 will contain the heading of GCV 2. After we retrieve these values, the x-positions of the GCVs (SS(1) and SS(2)) and the y-positions of the GCVs (SS(3) and SS(4)) can be calculated.

SAINT automatically advances simulated time for us. In defining our state equations, all we are required to do is recognize the fact that the current position of the GCV is equal to its last position plus the distance it has traveled in the appropriate direction during the last interval of time. Since SSL(•) represents the value of the state variable at the last time the state equation was updated and DTNOW represents the time since that update, the state equations given in Figure 36, when evaluated, will provide the values of the state variables that we require.

Modeling Using Derivatives of State Variables

An alternative to the representation of GCV flight presented in Figure 36 involves the use of derivative equations. SAINT uses a Runge-Kutta-England (RKE) algorithm to integrate the equations of subroutine STATE that are written in terms of derivative equations (8). To write the derivative equation representation of GCV flight, we use the following SAINT COMMON variables:

DD(•)	The value of the derivative of state variable I at time TNOW.
DDL(•)	The value of the derivative of state variable I at time TTLAS.

When equations are written using DD(•) variables, the RKE algorithm integrates them to obtain the values of the corresponding SS(•) variables. The RKE algorithm is capable of solving a set of simultaneous first order ordinary differential equations. Thus, if we code the derivatives of our GCV positions in subroutine STATE, we obtain the same values of SS(•) as we did in our previous representation of the system.

The derivatives of the x and y positions of the GCVs are the x and y velocities. Thus, the x-velocity of a GCV is represented by the following equation:

$$x' = \cos (h) * v$$

Similarly, the equation for the y-velocity of a GCV is represented by:

$$y' = \cos (h) *V$$

Now, if we wish to model GCV flight in terms of derivative equations, we define DD(1) as the x-velocity of GCV 1, DD(2) as the x-velocity of GCV 2, DD(3) as the y-velocity of GCV 1, and DD(4) as the y-velocity of GCV 2. Subroutine STATE for this representation of our GCV system is given in Figure 37.

When DD(•) equations are included in subroutine STATE, SAINT automatically integrates their values to determine the values of the corresponding SS(•) variables. Thus, the values of the SS(•) variables at time t will be the same as those in the difference equation representation of GCV flight, where SS(1) is the x-position of GCV 1, SS(2) is the x-position of GCV 2, SS(3) is the y-position of GCV 1, and SS(4) is the y-position of GCV 2.

Alternative State Variable Representations

We now have two alternative methods for representing state variables. If we use SS(•) equations, we must perform the computation of the integration of their values over time ourselves. If we use DD(•) equations, the SAINT RKE integration package performs the integration automatically for us. Further, SAINT allows us to include both SS(•) and DD(•) variables in subroutine STATE. In the GCV example, it is relatively easy to perform the integration since we have a system of linear equations. However, for more complex systems, it is often necessary to allow SAINT to perform the integration using its sophisticated RKE integration scheme, since this scheme arrives at more exact values of the SS(•) variables than we would calculate. With these capabilities, SAINT provides us with the ability to model highly complex state variable configurations.

```
      SUBROUTINE STATE
C
      COMMON /COM17/ SS(100),SSL(100),DD(100),DDL(100),LLRES(100,2)
C
C**** RETRIEVE THE HEADINGS AND VELOCITIES OF THE GCVS
C
      CALL GETSA(1,V1)
      CALL GETSA(2,V2)
      CALL GETSA(3,H1)
      CALL GETSA(4,H2)
C
C**** COMPUTE THE DERIVATIVES OF THE GCV COORDINATES
C
      DD(1)=COS(H1)*V1
      DD(2)=COS(H2)*V2
      DD(3)=SIN(H1)*V1
      DD(4)=SIN(H2)*V2
C
      RETURN
      END
```

Figure 37. Subroutine STATE Using DD(·) Equations.

Time Advance Mechanism

Before leaving the discussion of state variables, we must describe the means by which SAINT advances simulated time. In SAINT, simulation time is advanced in steps. The amount by which simulated time is advanced at each step is DTNOW. The value of DTNOW is variable, and depends on the type of equations modeled and the values of specific variables at the current point in time. If no state variables are involved, simulated time is advanced from one task completion to the next. Since task completions are scheduled to occur at some future point in time, these event times are known. However, when state variables are involved, time is incremented in steps between scheduled task completions for the purpose of updating the values of the state variables.

SAINT will advance simulated time by the maximum step size allowed until the end of the simulation unless conditions warrant a reduced step size. The selection of an appropriate step size is made automatically by SAINT. This selection depends on such factors as the integration accuracy required (user specified), the minimum and maximum step sizes allowed (user specified), and the time of the next scheduled task completion. In this manner, the SAINT time advance mechanism allows for all model contingencies, choosing a step size appropriate to the requirements of the particular model.

Writing Subroutine STATE

Subroutine STATE is written to compute the current value of each state variable or its derivative. Subroutine STATE is called frequently, especially if there are derivative equations in the model, and therefore should contain only essential code. It is most efficient if the state variable equations are numbered sequentially starting at 1.

There are several policies which are to be adhered to in writing subroutine STATE. By definition, the largest subscript for a derivative (DD(•)) equation is NNEQD. This value must be defined on input. Therefore, equations defining the rate of change of SS(I); i.e., an equation for DD(I), must satisfy the expression $I \leq$ NNEQD.

Also defined on input is NNEQS, the number of state variable equations written in terms of SS(•). Thus, the largest subscript I that can be used for SS(•) equations is NNEQD+NNEQS, or NNEQT. An equation defining SS(I) must satisfy the expression NNEQD$<I\leq$NNEQT.

To explain how this numbering system works, assume that we have 5 DD(•) equations and 3 SS(•) equations. On input, we would specify NNEQD=5 and NNEQS=3. SAINT would then calculate NNEQT as NNEQD+NNEQS=8. Then, in subroutine STATE, subscripts 1-5 would be used for DD(•) equations, while subscripts 6-8 would be used for the SS(•) equations. In general, if we have M DD(•) equations and N SS(•) equations, then the most efficient specifications would be NNEQD=M, NNEQS=N, and NNEQT=M+N. For an equations defining DD(I), $I \leq$ NNEQD. For an equation defining SS(I), NNEQD$<I\leq$NNEQT.

Either NNEQD or NNEQS can be 0. Further, since it is most efficient to have the DD(•) variables numbered sequentially, NNEQD is often referred to as the number of defining equations written for DD(•) variables. Similarly, NNEQS is often referred to as the number of defining equations written for SS(•) variables.

The preceding discussion delineates the numbering procedure to be used for state variables. The order in which the equations are written in subroutine STATE is left up to us, i.e., a statement defining DD(5) can precede a statement defining DD(3). Because SAINT does not change the execution sequence of state variable equations, correct sequencing of state and derivative equations is our responsibility. If the defining equations for DD(•) do not involve other DD(•) variables, then any order is permitted and the RKE integration procedure simultaneously solves for all the DD(•) and the corresponding SS(•) variable values.

If the defining equations for DD(•) do involve other DD(•) variables, then ordering becomes important, i.e., if DD(3) is a function of DD(5), then the equations for DD(5) must precede the equation for DD(3) according to standard FORTRAN conventions. Thus, if there are simultaneous equations involving DD(•) variables, we must develop an algorithm for solving this set of equations. This is also the case when a set of simultaneous DD(•) equations is included in subroutine STATE.

In subroutine STATE, the equations for DD(•) and SS(•) can be written in a variety of forms. The equations

DD(M) = RATE, and

SS(M) = SSL(M) + DTNOW * RATE

are essentially equivalent. Equations which define both DD(M) and SS(M) at the same time instant are not permitted since they would either be redundant or conflicting. When an equation for DD(M) is written, values of SS(M) are obtained through the RKE integration routine contained within SAINT. Values of DDL(M) and SSL(M) are automatically maintained. The step size is automatically determined to meet specified accuracy requirements on the computation of SS(M). When the equation is written for SS(M), only SSL(M) is maintained. In this case, the step size for updating SS(M), DTNOW, is maintained at the maximum value specified by the user unless conditions warrant a reduced step size.

The form of the equations for DD(•) and SS(•) is limited only by the FORTRAN statement types. This allows for a great deal of flexibility in defining the state variables of a model. In fact, state variable descriptions can be conditioned on system status. For example, if a rate is to change after 100 time units have elapsed, the code in subroutine STATE could be written in the following manner:

```
IF(TNOW.LT.100)DD(M) = SS(3) * SS(4) +SS(5)

IF(TNOW.GE.100)DD(M) = SS(3)
```

Since the use of the two equations is mutually exclusive, SAINT does not consider them to be redundant or conflicting. In this manner, SAINT provides the capability of altering state variable equations forms as a function of system status. Other alternatives to the formulation presented above will be discussed later.

Initializing State Variables

When state variables are included in a SAINT model, we are also required to write subroutine INTLC to initialize the values of the state variables defined by equations in subroutine STATE. For our GCV example, we need to specify the initial (launch) positions of the 2 GCVs. In other words, we must assign values to SS(1), SS(2), SS(3) and SS(4) in subroutine INTLC. These represent the positions of the GCVs at time zero. If we assume that the initial position of GCV 1 is (3,4) and the initial position of GCV 2 is (7,2), subroutine INTLC would be coded as in Figure 38. SAINT COMMON block COM17 is included in subroutine INTLC since we are defining the values of SS(•) variables.

Only the SS(•) variables that have initial values other than zero need be initialized in subroutine INTLC, since SAINT automatically assigns the value of zero to all SS(•) variables

```
          SUBROUTINE INTLC
    C
          COMMON /COM17/ SS(100),SSL(100),DD(100),DDL(100),LLSVR(100,2)
    C
    C**** INITIALIZE LAUNCH POSITIONS OF GCVs.
    C
          SS(1)=3.
          SS(2)=7.
          SS(3)=4.
          SS(4)=2.
    C
          RETURN
          END
```

Figure 38. Subroutine INTLC for the SAINT Model of the GCV System.

prior to execution of subroutine INTLC. Further, the initial values of SSL(·) variables are automatically set to the initial values of SS(·) variables by SAINT following the call to subroutine INTLC. Finally, even if we use DD(·) equations to define our state variables, the SS(·) equivalents must be assigned initial values in subroutine INTLC.

State Variable Output

We may obtain three types of state variable output from SAINT. The output may consist of plots of the values of state variables versus time, tables of the values of state variables versus time, and/or time-integrated statistical summaries of state variable values. All state variable output is prepared for the individual iterations that we specify on input. We may specify any combination of the three types of output for each state variable included in a model.

If we desire plots of the values of state variables, we simply specify the indices of the state variables to be plotted, the label to be used for each plotted state variable, and the time interval between consecutive recordings of their values. The appearance of the plotted output provided by SAINT for those variables we specify is very similar to other line printer plots such as those produced by GASP IV (8).

We have three options in the specification of the maximum (minimum) ordinate value for each variable to be plotted. We may specify that the maximum (minimum) ordinate value is to be the maximum (minimum) observed value during the simulation, or that the maximum (minimum) ordinate value is to be that specified on input, or that the maximum (minimum) ordinate value is to be rounded upward (downward) to the nearest specified value. These capabilities give us a wide range of options in designing a plot.

A second form of statistical output that we can obtain from SAINT is a table. We specify on input, in much the same manner as for a plot, the index of each state variable to be tabled, the label to be used for each tabled variable, as well as the time interval between consecutive recordings of their values. SAINT automatically records each value, notes the time at which it was recorded, and outputs this information at the end of each iteration specified. Further, SAINT automatically computes the maximum and minimum values observed for each variable in the table.

The third form of output associated with state variables is a time-integrated statistical summary of the values of the variables. On input, we specify the indices of the state variables for which SAINT is to collect statistics. For each of these variables, SAINT automatically integrates their values over time. At the end of each iteration for which state variable statistics are requested, SAINT provides a statistical report which includes the average value of each variable, as well as the minimum and maximum values observed during the iteration.

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, re-read the section.

1. Variables whose values change continuously over time are called state variables.

2. State variables are described by algebraic, differential, or difference equations that govern their time-dependent behavior.

3. State variable equations must be coded in subroutine STATE in FORTRAN or a compatible language using special SAINT COMMON variables.

4. A table which includes the state variable number, label, defining equation in subroutine STATE, and initial value portrays the state variable model component on the network drawing.

5. Algebraic or difference state variable equations must be of the form "SS(I)=", where I is the state variable number.

6. SAINT labeled COMMON block COM17 must be included in subroutine STATE.

7. SAINT automatically advances simulated time, accounting for task completions and state variable updating.

8. Differential state variable equations must be of the form "DD(I)=", where I is the state variable number.

9. SAINT automatically integrates DD (•) equations to determine the values of the corresponding SS (•) variables.

SECTION XI

SWITCHES

In Section X, we modeled the flight of the GCVs using state variables. They fly in a linear path from their launch point along prescribed headings. However, even though we have a task whose completion represents the launch of a GCV, the GCVs are actually flying beginning at time 0. For this reason, SAINT incorporates the capability for communication between the task-oriented and state variable components of a model.

We also presented an alternative for specifying the equations in subroutine STATE in which we based the use of individual state variable equations on the value of TNOW, the current time. If we knew the exact time of launch of each of the GCVs, we could use this same technique in our model to launch the GCVs at the correct time. However, since the tasks representing GCV launch have performance times that are based on probability distributions, we do not know *a priori* the exact times that the tasks will be completed (the GCVs will be launched). Thus, we need a mechanism by which we can inform the state variable component of the model that the GCVs have been launched. The vector IS(·) provides that mechanism.

Using Switches

The vector IS(·) is included in SAINT COMMON. The subscript of the array is the switch number. Initially, the values of the IS(·) vector are assumed to be off, i.e., IS(·) = 0. These values can be reset in subroutine INTLC or at the completion of any task in the network. If we code subroutine STATE to be dependent on switch values, and we turn on the appropriate switch at the time of each GCV launch, then we can provide the interaction necessary to initiate GCV flight, as represented by the state variable component of the model, at the correct time.

Switches for the GCV Model

Let us define IS(1) as a switch which is equal to 1 if GCV 1 has been launched and equal to 0 if GCV has not been launched. IS(2) is defined similarly for GCV 2. Thus, at the completion of task 1, switch 1 will be set of 1. At the completion of task 2, switch 2 will be set to 1.

Assigning Switch Values at Tasks

To simplify the model, assume that the required resource and system attribute values have been assigned by means of initial attribute specifications instead of being assigned at the tasks as shown previously. Under this framework, Figure 39 illustrates the required switching described above. The task description code for switching information is SWIT. We specify the switches to be reset and their new values on the right-hand side of the SWIT row. Thus, upon completion of task 1, switch 1 is set to 1 (IS,1=1). In the same fashion, switch 2 is set to 1 (IS,2=1) at the completion of task 2.

Coding Subroutine STATE Using Switches

Figure 39 displays the representation of the interaction on the task-oriented model component, but how do we include this interaction in subroutine STATE? Let us assume that we are using an SS(·) representation for the state variables in our model. In this case, we must insure that the state variable equations describing GCV position are not evaluated until the switches for the GCVs are equal to 1. Subroutine STATE for this situation is illustrated in Figure 40. The only other addition to subroutine STATE is the inclusion of COMMON block COM18, which contains the IS(·) vector.

Summary of the GCV System

To recap, we now have a SAINT model of the following GCV system. Operator 1 launches GCV 1 at task 1 and GCV 2 at task 2. These tasks cause the appropriate switches to be set to 1. When the switches are 1, the GCVs are set in motion through subroutine STATE. They fly along the heading and velocity specified as system attributes. Now, it seems that we have developed a reasonably accurate representation of our system. However, we note that at task 4 we record the status of the GCV, yet we still have no mechanism for transferring information from the state variable component to the task-oriented component of the model. The method by which we can accomplish this transfer of information is discussed in the next section.

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, re-read this section.

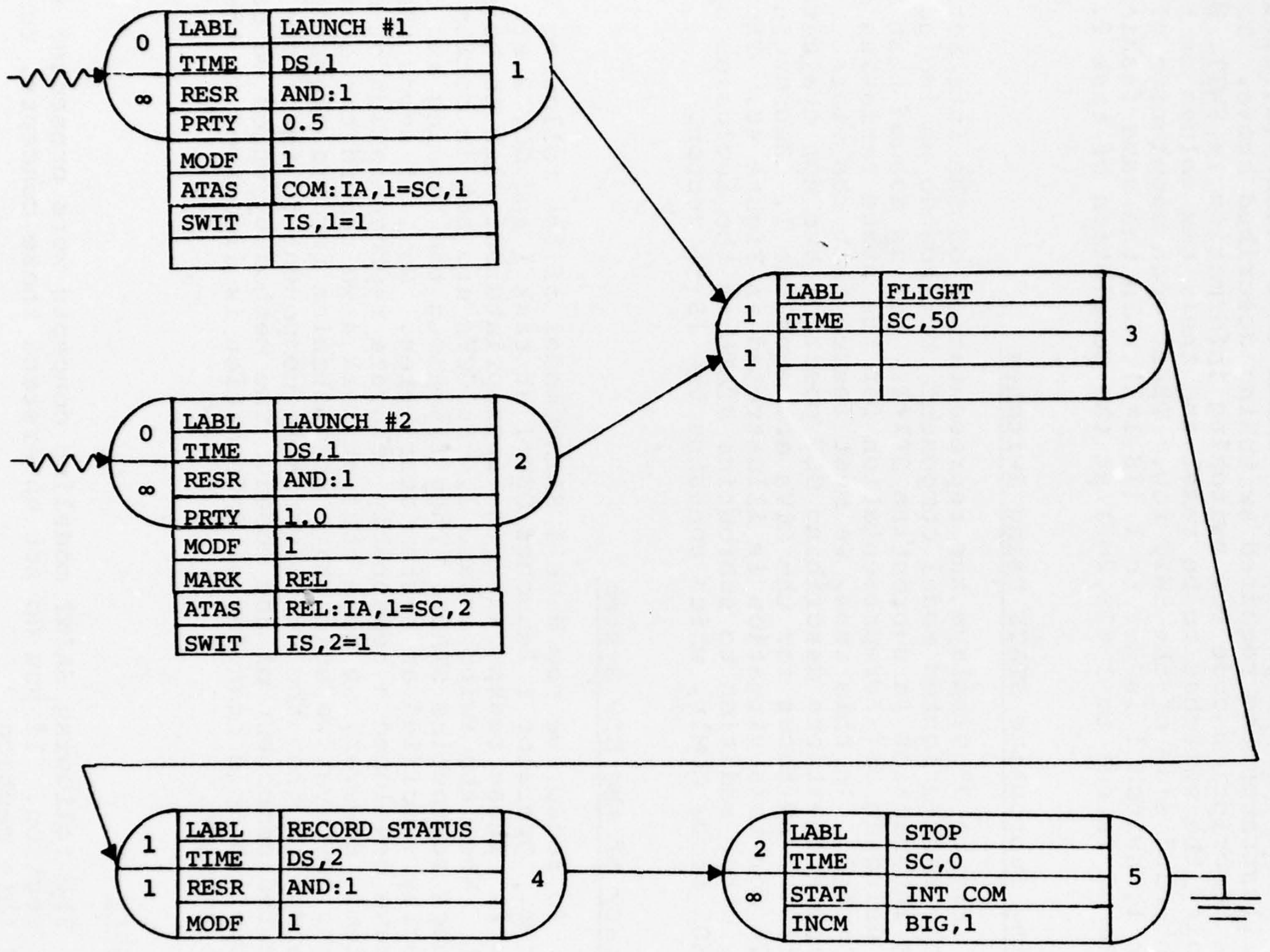


Figure 39. SAINT Model Illustrating Setting Switch Values at Tasks.

```
      SUBROUTINE STATE
C
      COMMON /COM17/ SS(100),SSL(100),DD(100),DDL(100),LLSVR(100,2)
      COMMON /COM18/ IS(20),NABAD(300),YABAR(600)
C
C**** RETRIEVE VELOCITIES AND HEADINGS FROM SYSTEM ATTRIBUTES.
C
      CALL GETSA(1,V1)
      CALL GETSA(2,V2)
      CALL GETSA(3,H1)
      CALL GETSA(4,H2)
C
C**** COMPUTE GCV X AND Y COORDINATES IF GCV IS FLYING.
C
      IF(IS(1).EQ.1) SS(1)=SSL(1)+COS(H1)*V1*DTNOW
      IF(IS(2).EQ.1) SS(2)=SSL(2)+COS(H2)*V2*DTNOW
      IF(IS(1).EQ.1) SS(3)=SSL(3)+SIN(H1)*V1*DTNOW
      IF(IS(2).EQ.1) SS(4)=SSL(4)+SIN(H2)*V2*DTNOW
C
      RETURN
      END
```

Figure 40. Subroutine STATE for SAINT Model in Figure 39.

1. The vector <u>IS(·)</u> is included in SAINT COMMON storage to provide a communications link between the state variable and task-oriented components of a SAINT model.

2. The subscript of the array IS(·) is the <u>switch</u> <u>number</u>.

3. Initially, the values of the IS(·) vector are assumed to be off, i.e., IS(·) = 0.

4. The values of the IS(·) vector can be reset in subroutine INTLC or at the completion of any task in the network.

5. The IS(·) vector can be used to control the operation of subroutine STATE, enabling the task-oriented component of the model to affect the computation of state variable values.

6. The task description code for assigning switch values at a task is <u>SWIT</u>.

7. The switches to be reset and their new values are specified on the right-hand side of the SWIT row.

8. SAINT COMMON block COM18 must be included in any user-written subprogram which employs the IS(·) vector.

SECTION XII

USER FUNCTIONS

Employing User Functions

A user function enables us to write our own FORTRAN function (function USERF) for generating attribute assignment values. At any task in the network, we can specify any attribute assignment using the function specification UF. Internally, this directs SAINT to call function USERF with the user function number as an argument. The required attribute values are computed in function USERF. SAINT automatically assigns these values to the attributes specified. In this manner, attribute assignment values can be computed as a function of any SAINT or user-defined variable.

To satisfy the requirements of our example, we will make an attribute assignment at task 4. In the assignment function, we specify "UF,I", where I represents the user function which applies for this task. Within the user function, we compute the necessary value and assign it to USERF. SAINT then assigns this value to the specified attribute. If we specify an attribute assignment at the completion of task 4 which sets information attribute 2 to the value computed in user function 1 (as is illustrated in Figure 41), then SAINT will automatically call USERF(1) to make the assignment. However, we are responsible for writing the user function code that computes the required assignment value. The standard form for writing function USERF is given in Figure 42. It must be written in FORTRAN or a compatible language and can incorporate any of the internal SAINT variables.

Function USERF

Since user functions allow us to compute attribute values as a function of SAINT or user-defined variables, they can provide a method of communication between the task-oriented and state variable components of a model. We will now examine the coding of function USERF for our GCV example. When USERF is called from task 4, we would like the user function to return the value of the deviation of the GCV from its flight path. Since we know that the GCV is designed to fly from coordinate (0,0) in an easterly direction, we can see that the Y-coordinate of the GCV will represent its deviation from its desired flight. Thus, we will assign the value of the Y-coordinate of the GCV to attribute 2. Since the Y-coordinate of the GCV is SS(3) for GCV 1 and SS(4) for GCV 2, we simply determine which GCV we are working on and set USERF to the appropriate SS(·) value.

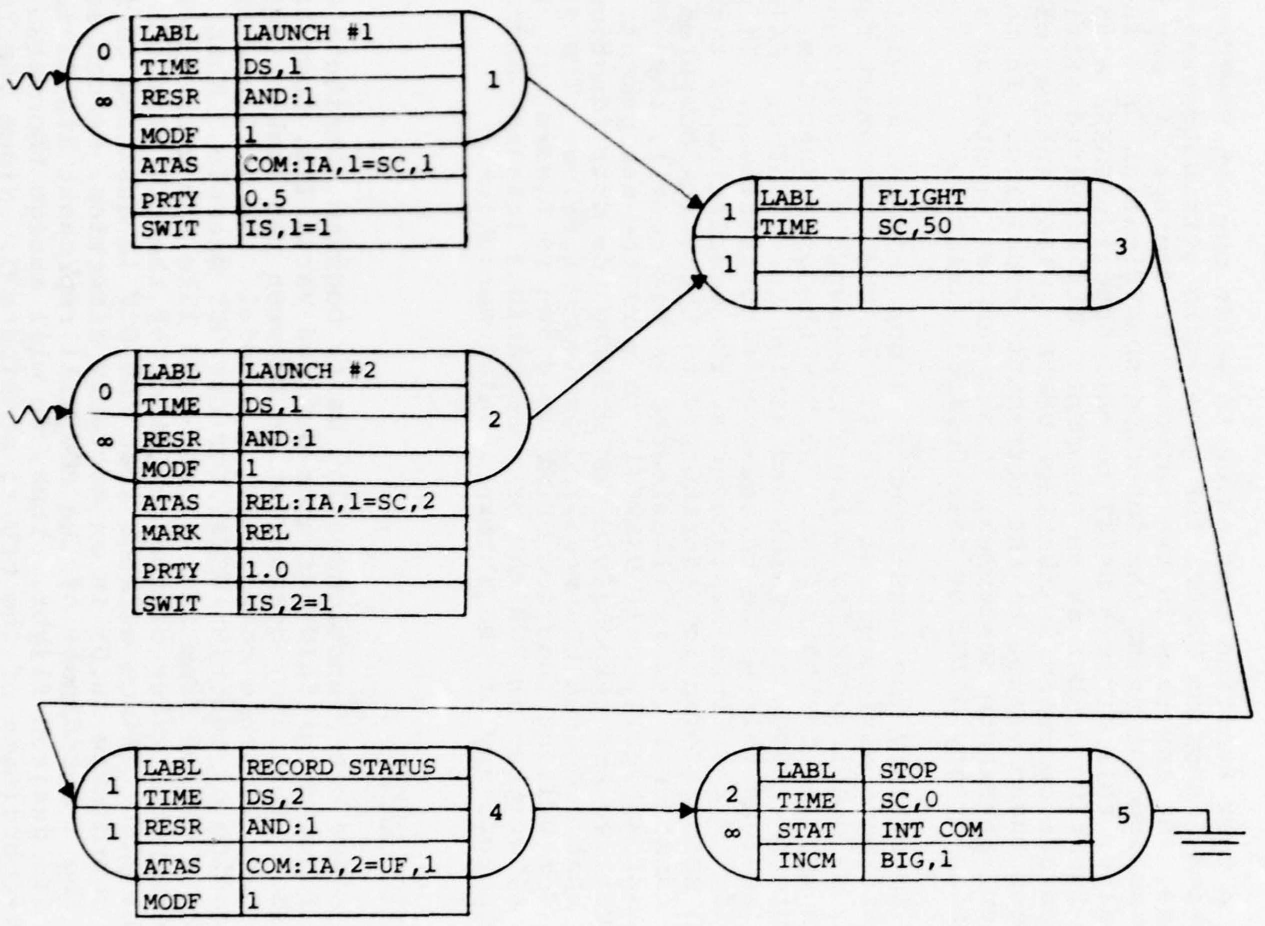


Figure 41. SAINT Model Illustrating an Assignment Through a User Function.

```
      FUNCTION USERF(IP)

      ***********
      COMMON CARDS (IF NECESSARY)
      ***********

      GO TO (10,20,30),IP

   10 *****FORTRAN Code for User Function 1.
      RETURN

   20 *****FORTRAN Code for User Function 2.
      RETURN

   30 *****FORTRAN Code for User Function 3.
      RETURN
      END
```

Figure 42. General Form of Function USERF.

Function USERF for our GCV system is given in Figure 43. Note that at the time function USERF is called, we must retrieve the value of information attribute 1 (the GCV number). To do so, we employ subroutine GETIA(NAT,VALUE), where NAT is the attribute number that we desired and VALUE is the value of this attribute returned by SAINT. Once we have retrieved the GCV number from the information attribute, we compute the appropriate SS(•) index and set USERF to the Y-coordinate of this GCV.

Navigation System Errors

Whenever user function 1 is called for our current GCV model, it will always return a value of 0. We have directed the GCVs in an easterly direction, and we know that there are no errors in flight. However, in reality, we realize that no GCV will fly without error. For this reason, we would like to model errors in the navigation systems onboard the GCVs.

We have set system attributes 3 and 4 to be the headings of GCVs 1 and 2, respectively. If at the time of the launch of these GCVs, we introduce a deviation in the values of the system attributes, we can model errors in GCV navigation systems. Let us assign the values of system attributes 3 and 4 through a user function of tasks 1 and 2, respectively. In this user function, we will take the initial value of the heading, as specified on input (which we have previously defined as 0) and add a normal deviate with a mean of 0 radians and a standard deviation of 0.001 radians. This process will introduce errors in GCV flight in our model.

Model of the GCV System with Navigation System Errors

The SAINT model of the GCV system with navigation system errors is shown in Figure 44. The FORTRAN code for function USERF, which assigns values to system attribute 3 at task 1, system attribute 4 at task 2, and information attribute 2 at task 4 appears in Figure 45. The navigation system errors are applied in user function 2 using the SAINT normal deviate generator, function RNORM. If we define distribution set 4 as a normal distribution with mean 0 and standard deviation 0.001, then we can call RNORM(4) to obtain a sample from this distribution. (Other random deviate generators available in SAINT are discussed in The SAINT User's Manual (1)). We compute this normal sample and set USERF to be the old value of the system attribute plus our random sample. With this new value in the system attribute for the heading of the GCV, subroutine STATE will compute GCV flight paths that deviate from those desired.

```
      FUNCTION USERF(IP)
C
      COMMON /COM17/ SS(100),SSL(100),DD(100),DDL(100),LLRES(100,2)
C
C**** DETERMINE THE GCV NUMBER
C
      CALL GETIA(1,VALUE)
      IGCV=VALUE
C
C**** DETERMINE STATE VARIABLE FOR Y-COORDINATE OF THIS GCV
C
      INDX=IGCV+2
C
C**** SET USERF TO THE Y-COORDINATE
C
      USERF=SS(INDX)
      RETURN
      END
```

Figure 43. Function USERF for SAINT Model in Figure 41.

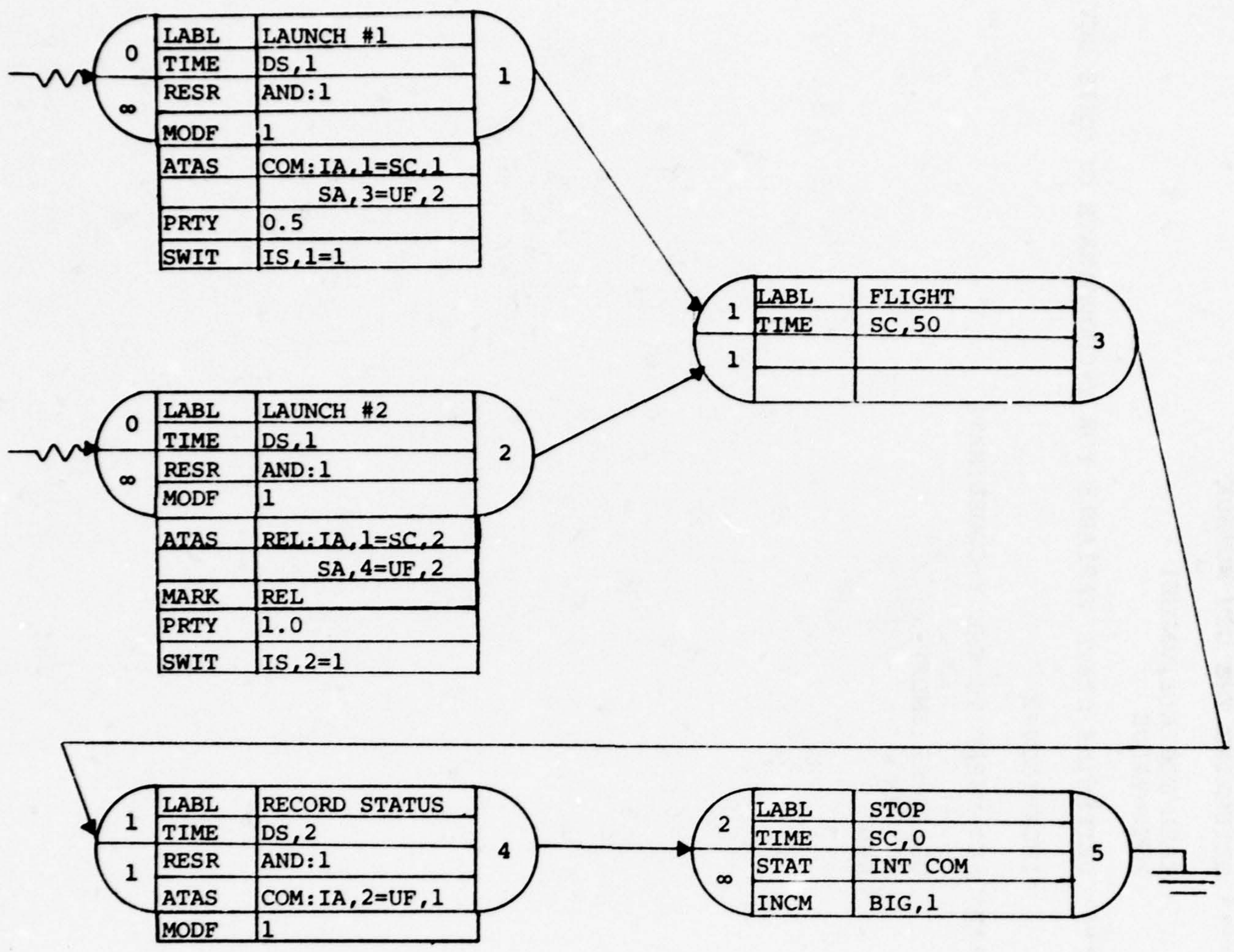


Figure 44. SAINT Model Illustrating System Attribute Assignments Through a User Function.

```
      FUNCTION USERF(IP)
C
      COMMON /COM17/ SS(100),SSL(100),DD(100),DDL(100),LLRES(100,2)
C
      GO TO (10,20),IP
C
C**** CODE FOR USER FUNCTION 1
C
   10 CALL GETIA(1,VALUE)
      IGCV=VALUE
      INDX=IGCV+2
      USERF=SS(INDX)
      RETURN
C
C**** CODE FOR USER FUNCTION 2
C
   20 CALL GETIA(1,VALUE)
      IGCV=VALUE
      INDX=IGCV+2
      CALL GETSA(INDX,VALUE)
      USERF=VALUE+RNORM(4)
      RETURN
      END
```

Figure 45. Function USERF for SAINT Model of Figure 44.

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, reread this section.

1. User functions are used to compute attribute assignment values as a function of any SAINT or user-defined variables.

2. User functions can be employed in any situation where attribute values are assigned.

3. The function type for a user function assignment is UF.

4. The parameter specification for a user function assignment is the user function number.

5. SAINT has no restrictions on the number of user functions included in a model.

6. Specifying UF,I as an attribute assignment causes Function USERF to be called with argument I.

7. Function USERF is user-written.

8. SAINT assigns the returned value of USERF to the specified attribute.

9. Subroutine GETIA is used in a user-written subprogram to retrieve the value of an information attribute.

10. Function RNORM can be used in a user-written subprogram to obtain a sample from a normal distribution.

SECTION XIII

BRANCHING ON ATTRIBUTES

In the preceding section, we have discussed how values from the state variable component of the model can be retrieved and assigned to attributes. Since SAINT is designed to model a complex system containing resources which perform tasks within an environment, it must allow the resources of the system to perform different tasks based on the status of the environment, i.e., the status of the state variables. For example, we would like to be able to determine the tasks that the operator must perform depending on the deviation of the GCVs from their desired flight paths.

SAINT Conditional Branching Capabilities

SAINT allows us to branch from one task in the network to another based on the values of the system, resource, or information attributes. The following list gives all the conditional branching conditions allowed by SAINT. Any one of these codes may be specified for a branch in the network which emanates from a conditional branching operation. Thus, the branch will be *taken only* if the condition specified is satisfied.

TVC	A task specified has been completed.
TVN	A task specified has not been completed.
TLV	The current time is less than or equal to a value.
TGV	The current time is greater than a value.
ALV	The value of an attribute is less than or equal to a specified value.
AGV	The value of an attribute is greater than a specified value.
TAC	The task whose number is found in an attribute has been completed.
TAN	The task whose number is found in an attribute has not been completed.
TLA	The current time is less than or equal to the value of an attribute

TGA	The current time is greater than the value of an attribute.
ALA	The value of one attribute is less than or equal to the value of another attribute.
AGA	The value of one attribute is greater than the value of another attribute.

The required parameters for each of these conditions is discussed in The SAINT User's Manual (1).

Branching Based on GCV Deviation

In our GCV system, we know that information attribute 2 will be set at task 4 to be the deviation of a GCV from its flight path. Let us assume that instead of being required to write down this deviation value, the operator must perform an operation that is dependent on this value. Suppose that at the time the operator determines the status of the GCV, he must decide whether the GCV requires a correction or whether it is close enough to its flight path to not require correction. Furthermore, let us assume that the operator will correct a GCV if its deviation is greater than 100 feet. Thus, we need to branch from task 4 depending on the value of information atttribute 2. If information attribute 2 is greater than 100, then we want to branch to a task which represents correcting the flight of the GCV. On the other hand, if the deviation of the GCV is less than or equal to 100 feet, the operator will not make a correction for this GCV. In the latter case, we will branch, as before, to task 5.

Correcting GCV Deviation

Let us define task 6 as a task which requires the operator to press a button to correct GCV flight. Pressing this button causes the computer to automatically correct the flight of this GCV (to head the GCV back towards its desired flight path). Since the operator is required to push the button, he is specified as being required for task 6. The performance time of task 6 will be defined in distribution set 5. The SAINT model illustrating the above situation is depicted in Figure 46. At task 4, we make a conditional-take first branching operation which selects the branch to task 6 if information attribute 2 is greater than 100. Otherwise, the branch to task 5 is taken. Thus, if the deviation is greater than 100 feet, task 6 will be released and the operator will be required to press the correction button that improves GCV flight. After the correction, the branch

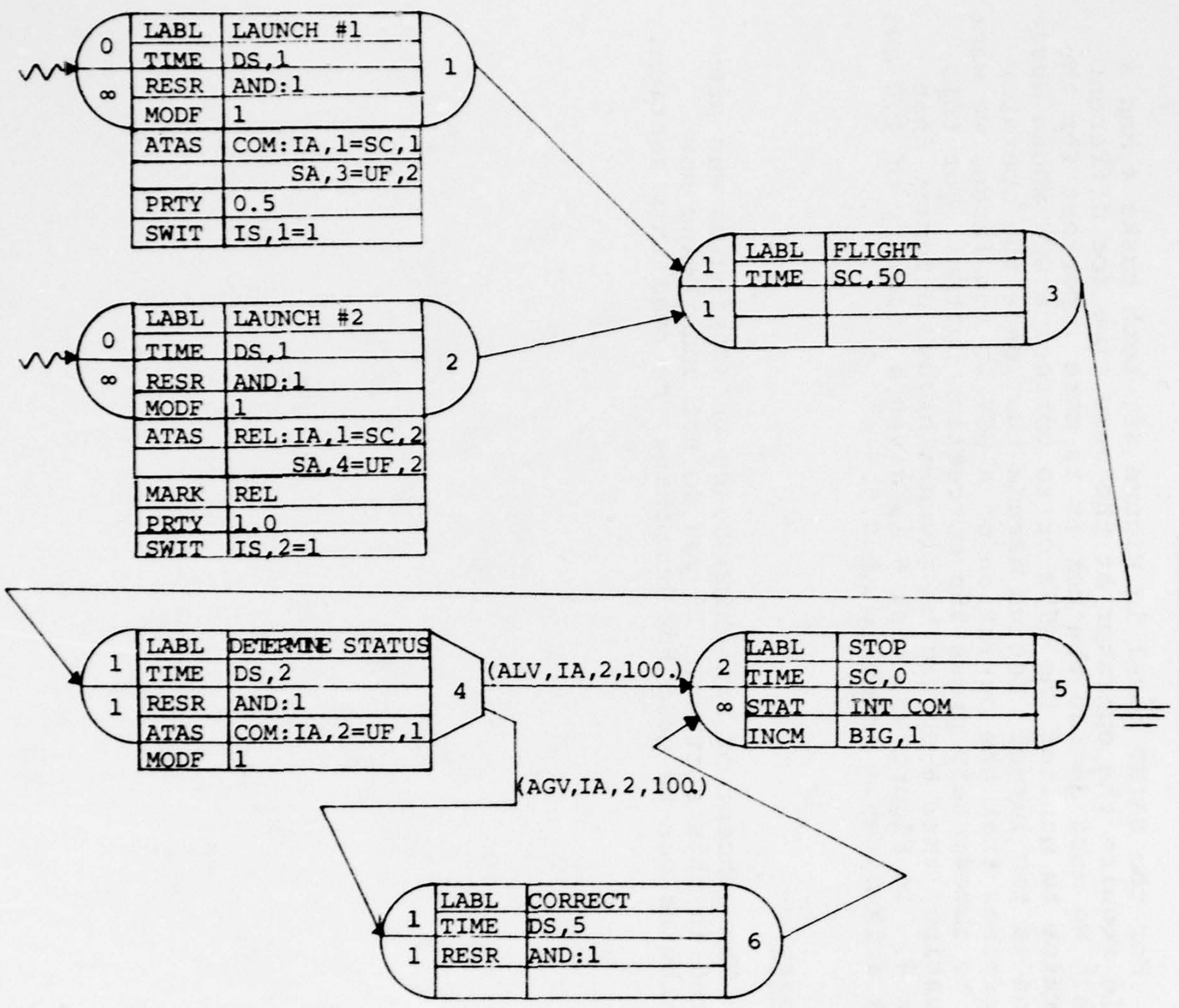


Figure 46. SAINT Model Illustrating Branching on Attributes.

to task 5 is taken deterministically, indicating that no further operations need to be performed on this GCV.

Further Use of Task Priorities

For the SAINT model in Figure 46, both tasks 4 and 6 could require the operator at the same time for different GCVs. We must decide whether it is more important for the operator to monitor the GCVs or to correct a GCV whose deviation is too large. Let us assume that once the operator determines that the deviation of a GCV is too large, we want him to immediately press the correction button. For this situation, task 6 should be given a higher priority than task 4. In Figure 46, task 6 is given a priority of 1.0 and task 4 is given a priority of 0.5.

Summary

The concept of branching based on attributes was presented in this section. If you do not understand how attributes can be used for branching, re-read this section.

SECTION XIV

DYNAMIC TASK PRIORITY

Our current GCV system is defined as follows: An operator launches two GCVs and allows them to fly for at least 50 time units. After a GCV has flown for at least 50 time units, the operator determines the status of the GCV and causes a heading change on the GCV if its deviation from the flight path is greater than 100 feet. After he has checked both GCVs, he is finished with his mission. To make the system more realistic, suppose that instead of allowing the GCVs to fly 50 time units and then check their status once, we assume that once the GCVs are launched, the operator continuously monitors the deviation status of the GCVs and makes heading corrections whenever necessary.

To set up this system, and to simplify the discussion, let us assume that we are no longer interested in computing the time it takes to perform the mission for different operators, i.e., we no longer need to be concerned with the moderator function specifications at tasks 1, 2, and 4, the mark at task 2, and the statistics collection at task 5.

Revised Model of the GCV System

Under the above framework, the SAINT model of our GCV system would appear as in Figure 47. Tasks 1 and 2 represent the launch of the GCVs as before. Since we no longer require the GCVs to fly 50 time units, we have deleted task 3. In its place, tasks 1 and 2 branch directly to task 4, where the operator determines the status of each GCV. If a GCV's deviation is less than 100 feet, no correction is required. In this case, we return the information packet for this GCV to task 4, indicating that additional monitoring is required. On the other hand, if the deviation is greater than 100 feet, task 6, which represents the operator pressing a button to correct the heading of the GCV, is released. After task 6 is completed, the information packet is returned to task 4, indicating that further monitoring is required.

Required Priorities of the Model

For this model, all tasks require resource 1. Thus, we must insure that the priority structure is input so that the model performs as we desire. Since we desire both GCVs to be launched before any monitoring operation is performed, we

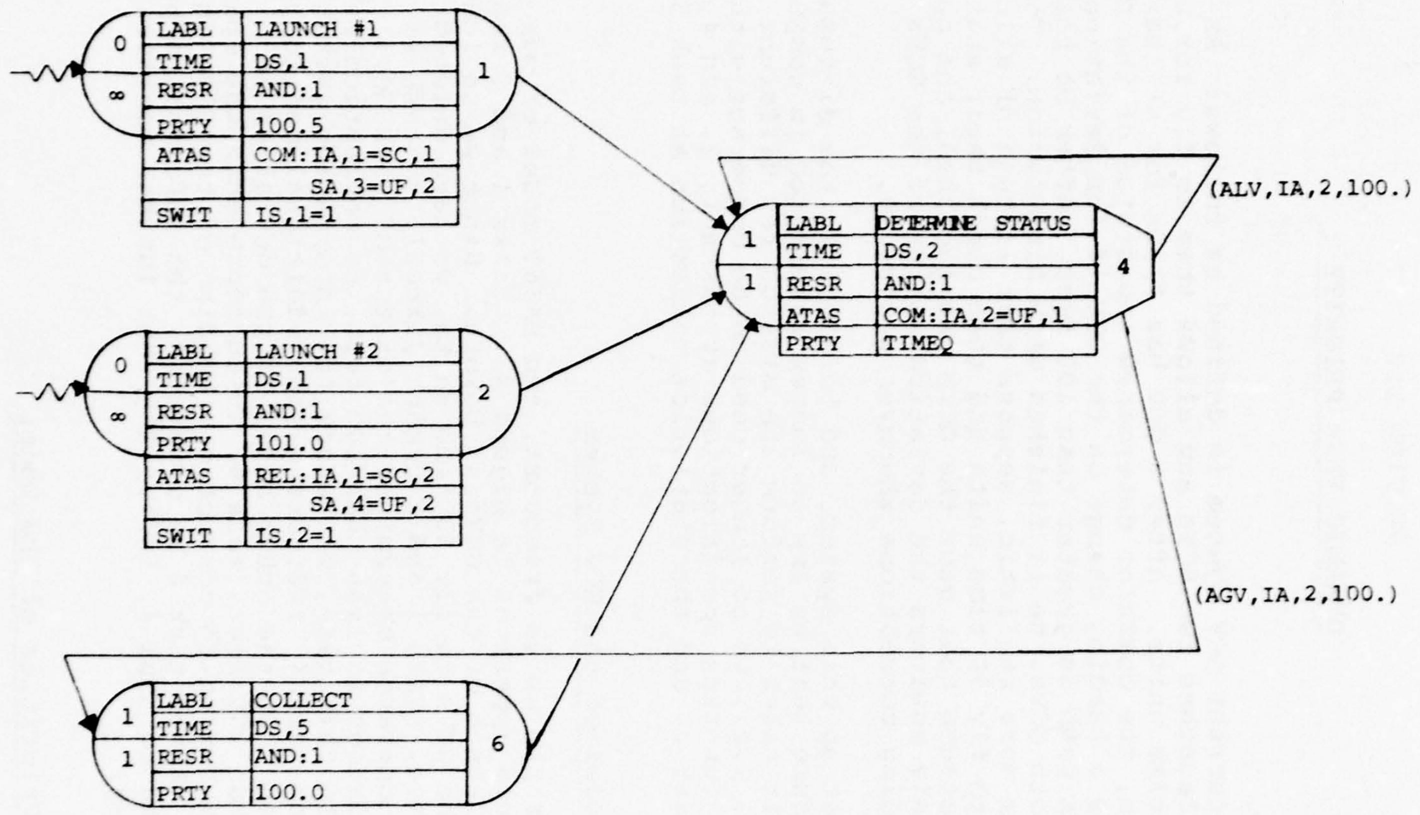


Figure 47. SAINT Model Illustrating Dynamic Priority.

want to set the priorities of tasks 1 and 2 greater than that of task 4. In addition, as we would like the operator to immediately correct the status of a GCV once it is found to be off course, we would like to set the priority of task 6 higher than that of task 4. To perform the above, we set the priority of task 1 to be 100.5, the priority of task 2 to be 101.0, and the priority of task 6 to be 100 (we still want GCV 2 to be launched prior to GCV 1).

Dynamic Priority Requirements

We will set the priorities of tasks 1, 2, and 6 to be greater than the priority of task 4. Whenever an attribute packet releases one of these tasks, it will be performed before task 4 is performed for any other attribute packet. Since all tasks require resource 1, he will not perform task 4 unless there is no higher priority task to be performed. Thus, whenever task 4 is to be performed, the operator will have to select one of the two GCVs to consider, since both will be waiting for status determination.

We would like to have control of which transaction the operator chooses. Thus, we would like to be able to set a priority that is based not only on the task, but also on the GCV whose status is to be determined. This can be accomplished through the use of the dynamic task priority capability in SAINT.

Function PRIOR

In addition to the capability for assigning priorities to tasks on input, we have the option of specifying a dynamic task priority. Before the scheduling of any task in the network, user-written function PRIOR is called by SAINT to assign the priority to all tasks awaiting scheduling (those tasks released but not yet started). If we do not write function PRIOR, the priorities used in ranking those tasks awaiting scheduling are those assigned on input.

Function PRIOR is accessed with a single argument, the task number for which a dynamic task priority is to be specified. We must write function PRIOR in FORTRAN or a compatible language. All standard language conventions must be observed. If we write function PRIOR, then we are totally responsible for setting the priorities of all tasks in the network through this function.

Using Dynamic Priorities

Since we write function PRIOR, all SAINT COMMON storage arrays are available. Thus, we may assign the dynamic task priority based on time, the task priority that was input, the value of an attribute, the status of a resource, etc. One piece of information that we may find particularly useful in the determination of dynamic task priorities is the amount of time that the task has been awaiting scheduling. A subprogram that retrieves this value has been included in SAINT. Function TIMEQ can be called from function PRIOR to compute the amount of time that the attribute packet associated with the task being considered has been waiting for resources so that it can be scheduled. Function TIMEQ is used with a dummy argument and returns the amount of time that this task has been awaiting scheduling.

Function PRIOR is called before the scheduling of any task in the network for any particular transaction. Thus, function PRIOR is called for each task that can be scheduled at the current time and for each transaction that resides at the task. In our case, when we have two transactions awaiting scheduling at task 4, function PRIOR will be called twice, once for each transaction.

Dynamic Priorities for the GCV System

Now consider our situation. If we assign the priorities based on the GCV number, i.e., we let the priority of the task associated with the transaction be equal to the GCV number, we would perform all operations on GCV 2 and none on GCV 1, since it would always have a larger priority.

Instead of using the above procedure, we would like to construct a dynamic priority scheme that will allow the operator to monitor the GCVs alternately. To accomplish this, we use function PRIOR in conjunction with function TIMEQ. If we set the priority of the transaction at task 4 to be equal to the time that the transaction has been awaiting scheduling, we can accomplish our objective. Whenever we are ready to schedule task 4, one of the transactions (GCVs) will have just arrived while the other transaction will have been waiting. The GCV that has been waiting will be the GCV that was not most recently monitored or corrected. Thus, if we assign our priorities properly, this transaction will be scheduled. As the simulation progresses, the result of this priority scheme will force the operator to alternately monitor the GCVs. Note that since we set the priority of the GCVs at task 4 to be the time awaiting scheduling, and since this time will be less than 100 seconds for all cases (assuming task times for tasks 4 and 6 are less than 10 seconds), the priorities of tasks 1, 2, and 6 will always be greater than the priority at task 4, which is what we originally desired.

Specifying Dynamic Priorities

To specify that we are writing function PRIOR and assigning dynamic priorities to all tasks in the network, we can specify on the task symbol the value given to the priority or the functional relationship that computes the priority. This information is placed in the right-hand side of the row that defines the priority (PRTY). For the situation described above, function PRIOR is given in Figure 48. Since function PRIOR is called for every task in the network, we must insure that the priorities of tasks 1, 2, and 6 are those that we input. If the task number is not equal to task 4, we branch and retrieve the priority value of the task as given on input. We use subroutine GETPR to retrieve the value. The arguments to this subroutine are the task number followed by the value retrieved. When we retrieve the value, we set PRIOR to this value. Thus, for tasks 1, 2, and 6, the dynamic priority is constant and is the value that we input for the task. On the other hand, for task 4, we simply set PRIOR equal to TIMEQ. This causes the priority to be the time that the transaction has been awaiting scheduling.

Task Scheduling

Up to this point, we have assumed that all tasks are scheduled based on the priority. However, this is not necessary in SAINT. While using the priority and/or dynamic priority capability in SAINT provides us with great modeling flexibility, we do have other options in determining how tasks are scheduled. On input, we can specify any of four rules that determine the sequence in which tasks are performed. The rules are as follows:

1. Low-value-first based on priority
2. High-value-first based on priority
3. First-in, first-out (FIFO)
4. Last-in, first-out (LIFO)

Only the first two ranking procedures utilize the task priority specified on input or computed dynamically. The last two ranking procedures ignore the task priority. In all the previous examples, we have assumed that the high-value-first based on the task priority rule has been used.

```
      FUNCTION PRIOR(ITASK)
C
C**** COMPUTE DYNAMIC PRIORITY OF TASK 4
C
      IF(ITASK.NE.4) GO TO 100
      PRIOR=TIMEQ(IDUM)
      RETURN
C
C**** COMPUTE PRIORITY OF OTHER TASKS
C
  100 CALL GETPR(ITASK,VALUE)
      PRIOR=VALUE
      RETURN
      END
```

Figure 48. Function PRIOR for SAINT Model of Figure 47.

However, for any case that you may come across, one of the above procedures will be the most appropriate and should be the one that you select.

SAINT also provides the capability for us to change, during the course of the simulation, the ranking procedure used in ranking those tasks awaiting scheduling. We can call subroutine QRANK to change the ranking procedure. Once the ranking procedure is changed, SAINT automatically reranks the file of tasks awaiting scheduling according to the new ranking procedure. The same four ranking procedures indicated above are available to us for use in conjunction with a call to subroutine QRANK. This subroutine is called with a single argument, whose value can be 1, 2, 3, or 4, designating one of the rules listed above.

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, re-read this section.

1. Dynamic task priorities can be used to assign task scheduling priorities to specific transactions at tasks.

2. Function PRIOR is the user-written subprogram in which dynamic priorities are calculated.

3. If function PRIOR is written by the user, it must assign the priorities to all tasks in the network.

4. Function TIMEQ can be called from function PRIOR to retrieve the time that a transaction has been awaiting scheduling.

5. Function TIMEQ can only be called from function PRIOR.

6. To specify dynamic priorities on the task symbol, the value given to the priority or the functional relationship that computes the priority is placed in the right-hand side of the PRTY row.

7. Subroutine GETPR can be used to retrieve the task scheduling priority assigned to a task on input.

8. A ranking procedure informs SAINT how to rank the file of tasks awaiting scheduling.

9. Task priorities are only used by SAINT when ranking is high value first (HVF) or low value first (LVF) based on priority.

10. The ranking procedure is defined by the user on input.

11. Other possible ranking procedures are FIFO and LIFO.

12. Subroutine QRANK can be called from any user-written subprogram to change the ranking procedure at any time during a simulation.

SECTION XV

STATE VARIABLE MONITORS

In our latest model, an operator launches two GCVs and continuously monitors and corrects their status. However, we notice that in this model we have no way of halting the simulation. Thus, if we coded up this model, it would proceed indefinitely. One possible method for stopping this model is to create a source task and a sink task in a disjoint network. We could have the performance of the source task require 0 time units and the performance of the sink task require 100 time units. When the sink task is completed, the simulation will end after 100 time units.

However, there are many cases where we would like to halt a simulation based on the status of the state variables. For example, in our GCV system, suppose that we want to halt the simulation only after the GCVs have flown a distance of 500,000 feet due east. Since a SAINT simulation can only be ended through a sink task, we must have a method for releasing tasks in the network as a function of state variable status.

Threshold Functions

Monitors provide the capability for monitoring specified state variables and comparing their values (either SS(•) or DD(•)) against prescribed threshold functions. A monitor will automatically search for the time that a state variable "crosses" the value of a threshold function, compute the values of the state variables at that time, and initiate specified actions associated with the monitor. Such actions may involve the setting of switch values and/or the signaling of tasks. The threshold function consists of a multiplicative constant, M; an SS(•) or DD(•) variable to be used in computing the threshold value, SS(V) or DD(V); an additive constant, C; a direction indicator, DIR; and a tolerance, TOL.

The monitored variable is compared against the threshold value of M*SS(V)+C (or M*DD(V)+C). The direction indicator, DIR, is important because we may wish to distinguish between crossing a threshold when the monitored variable is increasing and when the monitored variable is decreasing with respect to the threshold function. Three direction indicator settings are available:

UP	Monitoring for a crossing when the monitored variable is increasing with respect to the threshold function.
DOWN	Monitoring for a crossing when the monitored variable is decreasing with respect to the threshold function.
BOTH	Monitoring for a crossing in either direction.

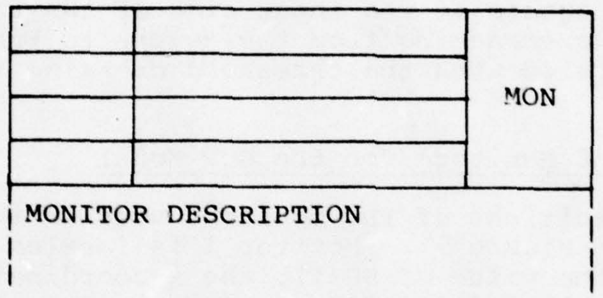
The tolerance, TOL, allows us to specify how close the SAINT program is to search for the exact time of crossing. The tolerance is normally computed as one or two percent of the threshold value. To illustrate the role of the tolerance specification, assume the threshold is 100 and the tolerance is 1. SAINT will attempt to identify the time of crossing so that the value of the monitored variable is between 100 and 101 if the UP direction is specified and between 100 and 99 if the DOWN directon is specified. If monitoring is in both directions, the former tolerance interval is used for crossings going up, and the latter tolerance interval is used for crossings going down.

Monitor Symbolism

The symbol which is used to represent a state variable monitor in a SAINT model is shown in Figure 49. Its design is similar to that provided for tasks. The left-hand side of this symbol contains the monitor description, and is divided into four rows. Each row represents a unique type of descriptive information, and is identified by one of the following description codes: LABL, MONF, MTAS, MSWT. The code LABL identifies a row containing the monitor label. The code MONF identifies a row containing the threshold function to be used. The code MTAS identifies a row containing the task(s) to be signaled as a result of a threshold crossing. The code MSWT identifies a row containing the switch(es) to be reset as a result of a threshold crossing. By using the description codes, only the information necessary to describe a monitor need be shown on the monitor symbol.

Use of Monitors for the GCV Model

For our GCV system, we have decided to end the simulation when both GCVs have flown east at least 500,000 feet. We would like to have a sink task signaled when each GCV has flown 500,000 feet east. This sink task will be required to be signaled twice before it can be started. Thus, two monitors will be included in the SAINT model. The first



MON = monitor number

Figure 49. Monitor Symbolism.

monitor will cause one predecessor completion of our sink task when GCV 1 has flown 500,000 feet east and the second monitor will cause another predecessor completion of the sink task when the second GCV has flown that far east. The SAINT model illustrating this system is depicted in Figure 50. Note that this model is identical to our previous model except that we have now included sink task 5. This task is labeled STOP, requires 0 time units to perform, and requires two predecessor completions for its release. These predecessor completions will come from monitors 1 and 2. To indicate this on the network, we place the monitor number in a square at the input side of the task and have a dotted line emanating from the square to the task that will be signaled when the threshold crossing is detected.

Definition of Monitors for the GCV Model

The definitions of the monitors used in the GCV model are given in Figure 51. Monitor 1 is labeled "STOP GCV 1". It detects the value of SS(1), the x-coordinate of GCV 1, crossing a value of 500,000 feet in the upwards directon, within a tolerance of 10 feet. No SS(•) or DD(•) variable is required for computing the threshold value. Since we have no switching as a result of monitor action, no switching is shown. The task signaled as a result of this threshold crossing is task 5 of the network depicted in Figure 50. Similarly, monitor number 2 is labeled "STOP GCV 2". It detects the value of SS(2), the x-coordinate of the second GCV, crossing the value of 500,000 feet in the upwards direction, within a tolerance of 10 feet; and also causes the signaling of task 5.

As designed, each monitor will be activated when its associated GCV crosses 500,000 feet east, and will signal task 5. Task 5 will require two predecessor completions before it can be released. Thus, the simulation will be ended only after both GCVs have flown 500,000 feet due east.

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, reread this section.

1. State variable monitors detect the time at which the value of a state variable "crosses" the value of a threshold function.

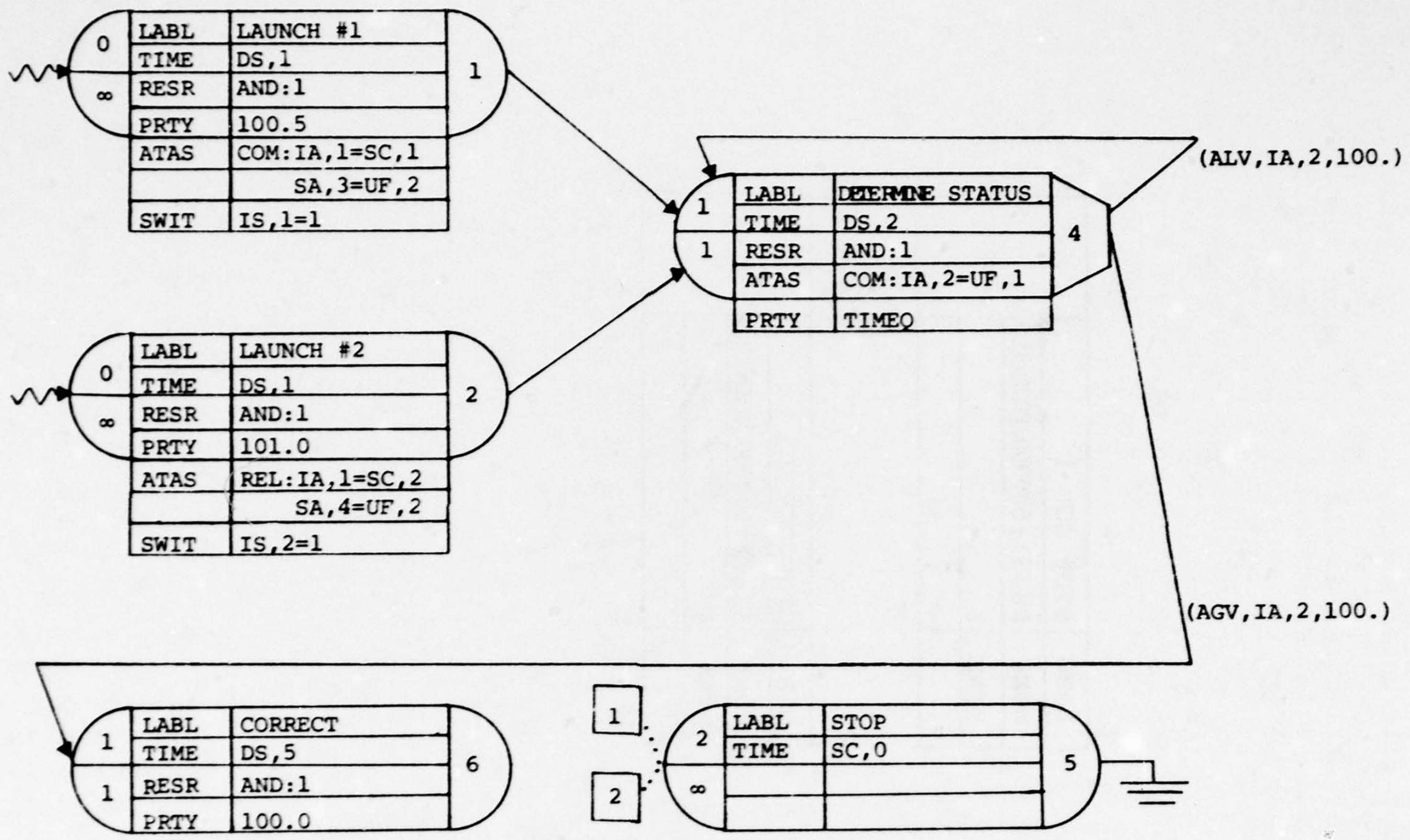


Figure 50. SAINT Model Illustrating Monitors.

LABL	STOP GCV 1	1
MONF	SS(3) ↑500000.(TOL=10.)	
MTAS	5	

LABL	STOP GCV 2	2
MONF	SS(4) ↑500000.(TOL=10.)	
MTAS	5	

Figure 51. Monitor Symbolism for SAINT Model of Figure 50.

2. The threshold function is defined by a multiplicative constant, M; an SS(•) or DD(•) variable to be used in computing the threshold value; an additive constant, C; a direction indicator, DIR; and a tolerance, TOL.

3. A monitor detects the time at which the value of a state variable (either SS(•) or DD(•)) "crosses" the value of M*SS(I)+C (or M*DD(I)+C) in the direction DIR and within the tolerance TOL.

4. The direction indicator distinguishes between crossings in which the state variable is increasing and decreasing with respect to the threshold function.

5. The tolerance informs SAINT how close it is to search for the exact time of crossing.

6. Once a monitor detects a threshold crossing, it may set switch values or signal tasks.

7. The monitor has its own distinct symbolism.

8. Task signaling as a result of monitor action is indicated on the SAINT network by a square containing the monitor number connected by a dotted line to the input side of the task to be signaled.

SECTION XVI

GCV HEADING CORRECTION

We now have a virtually complete model of our GCV system. The operator launches two GCVs and monitors and corrects their flight, when necessary, until both have flown 500,000 feet due east. In this model, we have accounted for all but one of the necessary interactions between the state variable and task-oriented components. What we have not accounted for is how, at task 6, the correction of the flight equations for the GCVs is made.

Upon completion of task 5 (Figure 50), we need to compute a new heading for the GCV whose flight path needs correction. This GCV is identified by the information packet associated with the task. If we are working on GCV 1, the new heading must be assigned to system attribute 3. If we are working on GCV 2, its new heading must be assigned to system attribute 4. However, since we do not know which GCV we will be working on _a priori_, we employ a user function ostensibly to compute a value to be assigned to information attribute 2, but in reality to determine which GCV we are working on and to make the necessary system attribute assignment. The information attribute assignment, employing user function 3, is shown in Figure 52.

Computation of Heading Correction

Let us assume that the final destination of the GCVs is the coordinate (500000,0). When an attribute packet arrives at task 6, we will know (through our state variables) the current position of the GCV. We also know the desired final destination of the GCV. Thus, in user function 3, we compute the heading that is required to take the GCV from its current position to its final destination, as shown in Figure 53. If the GCV is at coordinates (X,Y), then the heading that it must employ to reach coordinates (500000,0) is $-\alpha$ radians realtive to the X-axis (if (X,Y) is below the flight path, then the computation will be the same but $-\alpha$ will be a positive number of radians). In user function 3, we compute α as the arc sine of the y-coordinate of the GCV divided by 500,000 minus the x-coordinate of the GCV. The value of $-\alpha$ then represents the new heading of the GCV. However, since we know that GCV flight is not perfect (if we tell it to fly at a heading α, it will not fly exactly at that heading), we include the random deviate that represents the navigation system error.

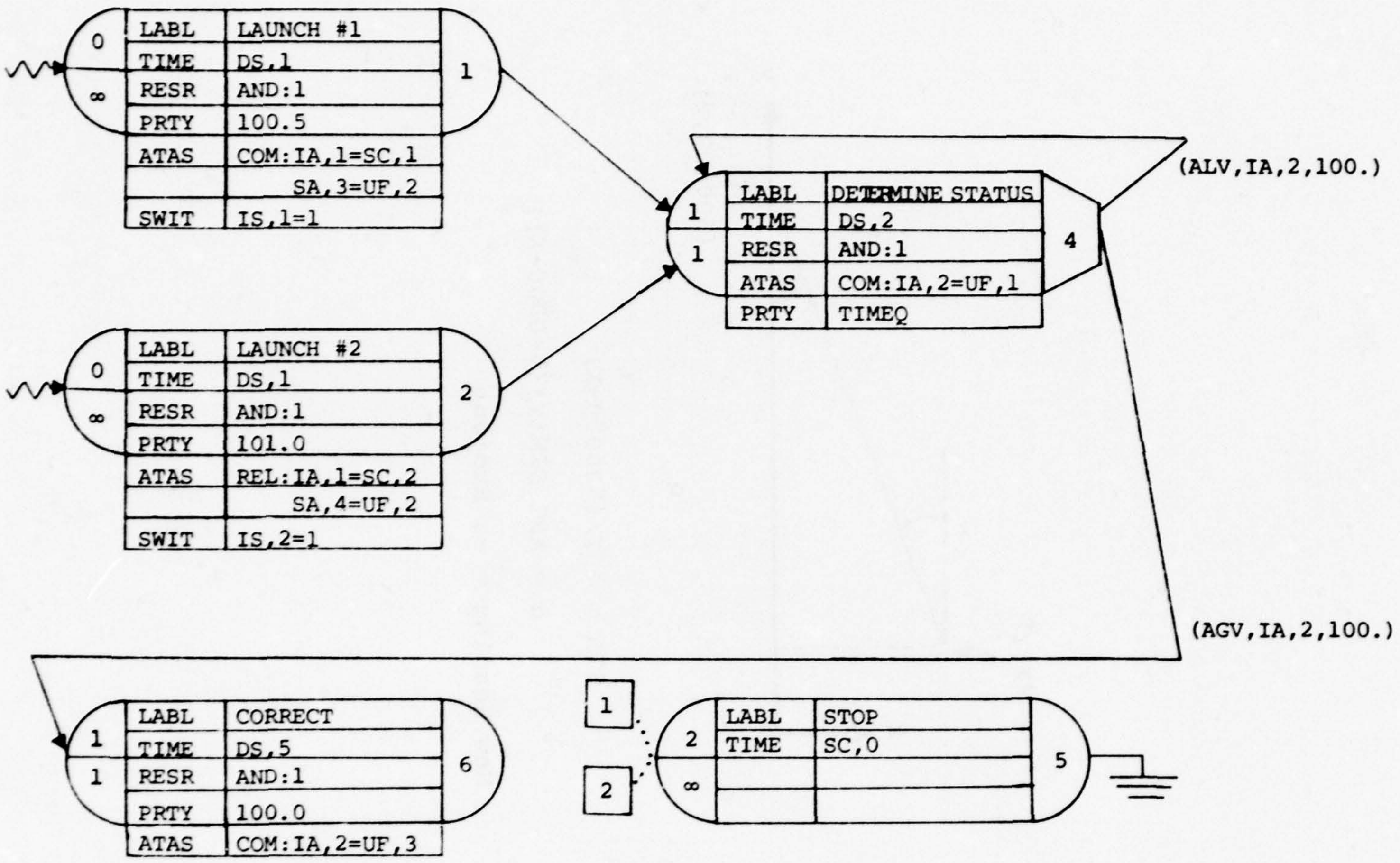
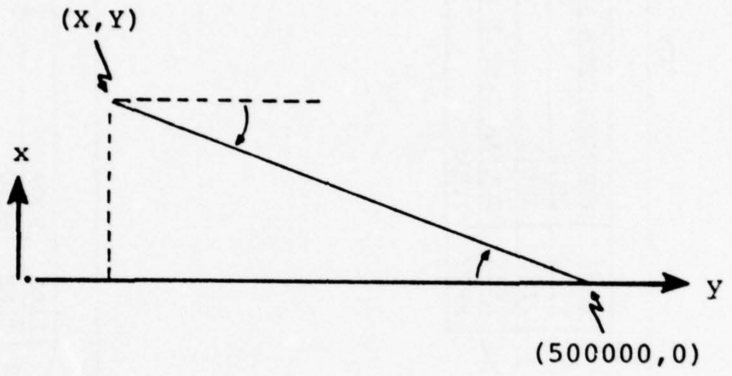


Figure 52. SAINT Model Illustrating Reassigning a Resource Attribute Through a User Function.



$$\text{SIN } \alpha = Y/(500000-X)$$

$$\alpha = \text{ARC SIN}(Y/(5-0000-X))$$

$$\text{New Heading} = -\alpha \text{ Radians}$$

Figure 53. GCV Deviation Calculation.

Coding of User Function 3

The user function for this model is shown in Figure 54. In user function 3 (starting at statement number 30), we first retrieve the information attribute containing the GCV number using subroutine GETIA (the arguments are the information attribute number and the returned value). Once we know the GCV number, we know the indices of the state variables which represent its position. We then compute $-\alpha$ and add a normal deviate from distribution set 4 (navigation system error) to this value to arrive at the new heading value. We then assign this new heading to the appropriate system attribute by using subroutine PUTSA with two arguments: the system attribute number and the value to be assigned to that system attribute.

Summary

This section describes the use of a user function to compute GCV heading corrections and to assign the new heading values to system attributes. If you do not understand the procedure involved, re-read this section.

```
      FUNCTION USERF(IP)
C
      COMMON /COM17/ SS(100),SSL(100),DD(100),DDL(100),LLRES(100,2)
C
      GO TO (10,20,30),IP
C
C**** CODE FOR USER FUNCTION 1
C
   10 CALL GETIA(1,VALUE)
      IGCV=VALUE
      INDX=IGCV+2
      USERF=SS(INDX)
      RETURN
C
C**** CODE FOR USER FUNCTION 2
C
   20 CALL GETIA(1,VALUE)
      IGCV=VALUE
      INDX=IGCV+2
      CALL GETSA(INDX,VALUE)
      USERF=VALUE+RNORM(4)
      RETURN
C
C**** CODE FOR USER FUNCTION 3
C
   30 CALL GETIA(1,VALUE)
      IGCV=VALUE
      X=SS(IGCV)
      Y=SS(IGCV+2)
      ALPHA=ASIN(Y/(500000-X))
      HEAD=-ALPHA+RNORM(4)
      CALL PUTSA(IGCV+2,HEAD)
      RETURN
      END
```

Figure 54. Function USERF for SAINT Model in Figure 52.

SECTION XVII

REGULATION

In the SAINT model of the GCV system discussed in the previous section, the operator monitors both GCVs until both have passed their 500,000 foot x-coordinate. After a GCV has passed its final destination, we no longer want to allow corrections on this GCV. One possible way to model this situation is to reset the state variables for this GCV so that when the operator checks the deviation of the GCV at task 4, the deviation of the GCV will always be 0. In essence, when task 5 (Figure 52) is signaled by monitor 1, indicating that GCV 1 has traveled 500,000 feet, we would like to set state variable SS(3) (the y-coordinate of GCV 1) to 0, representing a zero deviation from its due east heading. In addition, we will set switch 1 to 0 so that the GCV 1 flight equations are no longer updated. Similarly, when monitor 2 signals task 5, we would like to set the value SS(4) and the switch for this GCV (switch 2) to 0.

To do this, we decompose task 5 as shown in Figure 52 into two tasks (5 and 7), as shown in Figure 55. Now, instead of having a single sink task that requires two releases, we will define two sink tasks that each requires one release. We also will require two sink task completions to stop the simulation instead of one, as before. The first sink task, which will be signaled by monitor 1, will reset the value of switch 1 for GCV 1 and set SS(3), the deviation of GCV 1, to zero. Similarly, the second sink task will reset the switch and deviation values for the second GCV. However, before we define these tasks and present the model, we need to know how to change the value of a state variable when a task is completed.

Regulating State Variables at Tasks

Upon the completion of a task, the values of state variables can be regulated, i.e., instantaneously changed. For a task that causes the regulation, we specify the SS(•) variable affected by the regulation and a regulating function. The regulating function may be a constant or a linear function of any SS(•) or DD(•) variable. It is represented by a multiplicative constant, M; the subscript of the state variable to be used in calculating the regulation value (SS(V) or DD(V)); an additive constant, C; and a direction indicator (UP, DOWN, or TO). These parameters allow the regulation value, VAL, to take one of the following forms:

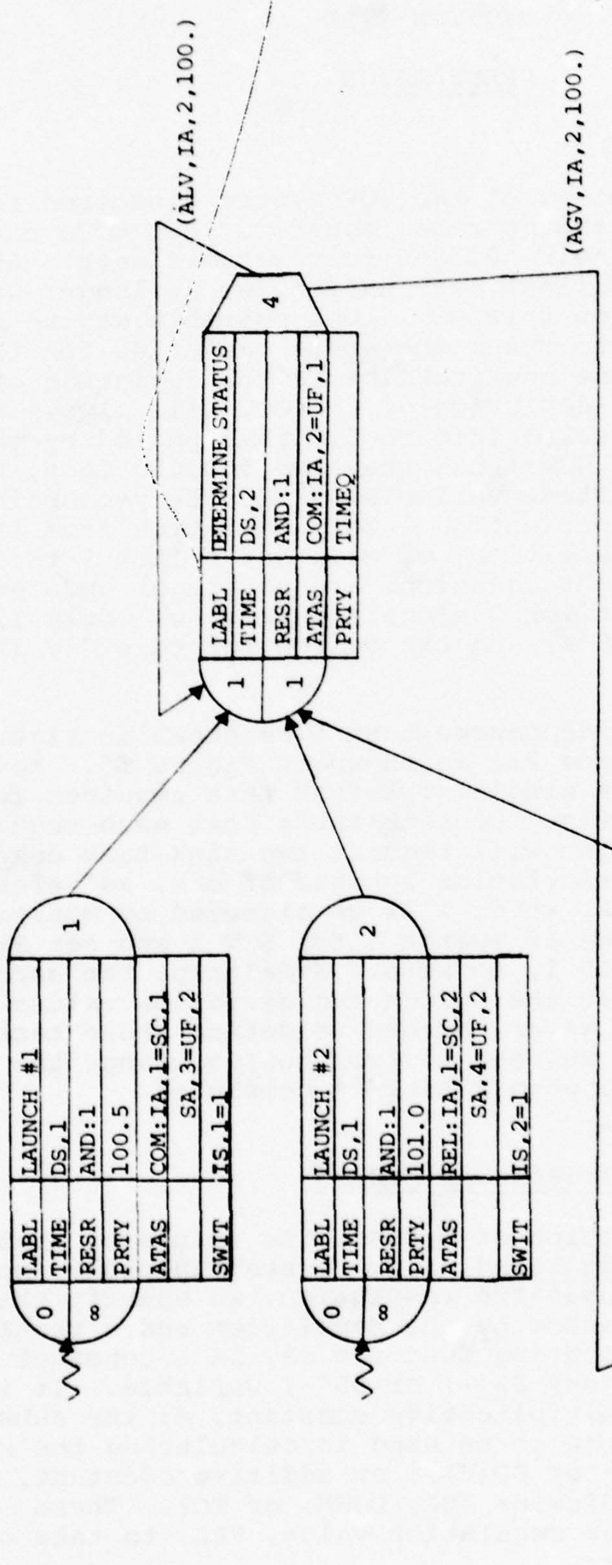


Figure 55. SAINT Model Illustrating Regulation.

VAL = M*SS(V) + C, or

VAL = M*DD(V) + C

When the direction indicator is UP, the state variable value will be increased by VAL. When it is DOWN, the state variable value is decreased by VAL. When it is TO, the state variable value is set equal to VAL. However, if the state variable being regulated is not a function of its past values, then the effect of the regulation is negated by the next call to subroutine STATE.

Regulation Symbolism

A regulation is identified on the task symbol by the task description code REGL. The right-hand side of the REGL row contains the regulating function. In Figure 55, task 5 is the sink task that is signaled by monitor 1 and task 7 is the sink task that is signaled by monitor 2. At task 5, we will set SS(3) = 0 and IS(1) = 0. For the latter, we use the switching capability discussed in Section XI. For the former, we need to specify a regulation. Thus, on the task symbol, we specify the code REGL and the required regulating function. Similarly, we set SS(4) = 0 and IS(2) = 0 upon the completion of task 7.

Summary of State Variable and Task-Oriented Component Interactions

With the capability of signaling tasks as a function of threshold crossings, setting switch values as a function of threshold crossings, regulating state variables as a function of task completions, setting switch values as a function of task completion, and assigning information attributes (through user functions) as a function of system status, complex systems that include both task and state variables can be modeled in SAINT. These capabilities can be used to represent a high degree of interaction between the state variable and the task-oriented model components.

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, re-read this section.

1. Upon the completion of a task, state variable values can be regulated (instantaneously changed).

2. Regulation causes the value of a specified state variable to be increased by, decreased by, or set equal to the value of a <u>regulating function</u>.

3. The regulating function may be a constant or a linear function of an SS(·) or DD(·) variable.

4. Multiple regulations may be performed based on a single task completion.

5. The task description code for regulation is <u>REGL</u>.

6. The regulating function is shown in the right-hand side of the REGL row on the task symbol.

SECTION XVIII

SAINT USER SUPPORT SUBPROGRAMS

In previous sections, we have identified and used a variety of user support subprograms provided by SAINT. This section summarizes the SAINT user support subprograms that can be called from our user-written subprograms to assist in SAINT modeling.

Attribute and Priority Manipulation

In the preceding sections, we have defined task priorities, system attributes, information attributes, and resource attributes. In addition, we have shown, for certain cases, how we can retrieve these values from subroutines USERF, MODRF, and PRIOR through the use of SAINT user support subprograms. For example, in Section X, we have written subroutine STATE with the use of subroutine GETSA, which retrieves the value of a system attribute. Besides retrieving values, SAINT also provides us with the capability for setting or resetting values of task priorities and attributes. For example, in Section XVI, we used subroutine PUTSA to reset the value of the system attribute representing GCV heading. The following list summarizes the subroutines that SAINT provides for manipulating attribute and priority values:

Subroutine	Function
GETIA(NAT,VALUE)	Retrieves VALUE as the value of information attribute number NAT of the information packet associated with the task now being processed.
PUTIA(NAT,VALUE)	Sets information attribute number NAT of the information packet associated with the task now being processed to VALUE.
GETRA(NRE,NAT,VALUE)	Retrieves VALUE as the value of resource attribute number NAT of resource NRE.
PUTRA(NRE,NAT,VALUE)	Sets resource attribute number NAT of resource NRE to VALUE.
GETSA(NAT,VALUE)	Retrieves VALUE number as the value of system attribute number NAT

Subroutine	Function
PUTSA(NAT,VALUE)	Sets system attribute number NAT to VALUE.
GETPR(ITASK,VALUE)	Retrieves VALUE as the task priority of task number ITASK.
PUTPR(ITASK,VALUE)	Sets the task priority of task number ITASK to VALUE.

Manipulating Task Scheduling and Task Statistics Information

SAINT allows us to retrieve and manipulate information associated with task scheduling and task statistics. For example, when we wrote function PRIOR in Section XIV, we employed function TIMEQ to retrieve the time that the task and associated attribute packet being considered had been residing in the file of tasks awaiting the assignment of resources. The support subprograms that enable us to manipulate scheduling and statistics information are listed below:

Subprogram	Function
Subroutine QRANK (IRANK)	Changes the ranking procedure used in ranking those tasks awaiting the assignment of resources to IRANK, where IRANK is 1 for low value first based on priority, 2 for high value first based on priority, 3 for FIFO, or 4 for LIFO.
Function TIMEQ(IDUM)	Retrieves the time that the task now being processed has been residing in the file of tasks awaiting the assignment of resources. This function is only accessible from frunction PRIOR. The argument IDUM is a dummy argument required by ANSI FORTRAN.
Function TMARK(IDUM)	Retrieves the mark time contained in the information packet associated with the task being processed. The argument IDUM is a dummy argument required by ANSI FORTRAN.

Using Random Deviates

If we find it necessary to generate random samples when we write subroutine MODRF, USERF, and PRIOR, we are allowed to call the SAINT random deviate generators to produce these values. For the majority of these calls, the random deviate generator is a function and we must include only the distribution set number as the argument. For example, if in subroutine MODRF we want to obtain a sample from a normal distribution, and the normal distribution is defined in distribution set 2, we would use the following statement: XX = RNORM(2). The following list identifies all the deviate generators contained in SAINT that can be used in this manner:

Function	Distribution Type
BETA	Beta or Beta-PERT
ERLNG	Erlang
GAMM	Gamma
RLOGN	Log Normal
RNORM	Normal
TRNGL	Triangular
UNFRM	Uniform
WEIBL	Weibull

In addition to the above, we can obtain a pseudo-random number between 0 and 1 through the use of function DRAND(IY), which generates a random number DRAND using the computer system's random number routine. This function is machine specific. The argument IY is the initial random number seed (ISEED). Also, we may use subroutine NPOSN(K,NPSSN), which generates a random deviate (NPSSN) from a Poisson distribution with parameters in distribution set K.

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, re-read the section.

1. Subroutine GETIA(NAT,VALUE) retrieves the values of information attributes.

2. Subroutine PUTIA(NAT,VALUE) resets the values of information attributes.

3. Subroutine GETRA(NRE,NAT,VALUE) retrieves the values of resource attributes.

4. Subroutine PUTRA(NRE,NAT,VALUE) resets the values of resource attributes.

5. Subroutine GETSA(NAT,VALUE) retrieves the values of system attributes.

6. Subroutine PUTSA(NAT,VALUE) resets the values of system attributes.

7. Subroutine GETPR(ITASK,VALUE) retrieves the values of task priorities.

8. Subroutine PUTPR(ITASK,VALUE) resets the values of task priorities.

9. Subroutine QRANK(IRANK) resets the ranking procedure used in ranking tasks awaiting scheduling.

10. Function TIMEQ(IDUM) retrieves the time that a task has been awaiting scheduling.

11. Function TMARK(IDUM) retrieves the mark time associated with a task awaiting scheduling.

12. Function BETA(NDIS) generates samples from Beta or Beta-PERT distributions.

13. Function ERLNG(NDIS) generates samples from Erlang distributions.

14. Function GAMM(NDIS) generates samples from Gamma distributions.

15. Function RLOGN(NDIS) generates samples from log normal distributions.

16. Function RNORM(NDIS) generates samples from normal distributions.

17. Function TRNGL(NDIS) generates samples from triangular distributions.

18. Function UNFRM(NDIS) generates samples from uniform distributions.

19. Function WEIBL(NDIS) generates samples from Weibull distributions.

20. Function DRAND(IY) generates a pseudo-random number between 0 and 1.

21. Subroutine NPOSN(K,NPSSN) generates samples from Poisson distributions.

SECTION XIX

USER-DEFINED TASK CHARACTERISTICS

With the capability of writing our own moderator functions, user functions, dynamic task priorities, etc., we may require task characteristics other than those defined by SAINT. For this reason, the capability has been included in SAINT for the specification of user-defined task characteristics. By using task characteristics in conjunction with moderator functions or other user-written subprograms, we are able to model the effect of different system conditions on task performance.

Using User-Defined Task Characteristics

How can we use task characteristics? Let us consider one possible use related to our GCV system. Suppose that we know that the performance of tasks deteriorates when the amount of light in the room is decreased and that the amount of deterioration depends on the task being performed. For example, we have defined tasks where the operator presses buttons to launch a GCV, obtains the status of the GCV, and corrects the status of the GCV. We can readily see that it may be difficult for the operator to press the appropriate buttons, when he is so required, when there is insufficient light in the room. However, the task of reading the CRT display will probably not be heavily dependent on the light level, since the CRT display produces its own light. Thus, launch and correction may be affected more by the lack of light than determining the status.

For the above example, we can define a task characteristic for each task which specifies the minimum light level for optimum task performance. Also, we can define a system attribute as the light level in the room. We can then write a moderator function which computes the difference between the required light level and the actual light level and increases task performance time as a function of this difference. If the light level is less than that required for a task, the task performance time of the task will be increased in some fashion to reflect the increased difficulty for the operator in performing his required action.

Retrieving and Resetting User-Defined Task Characteristic Values

For each task in a SAINT network, we may specify a set of task characteristics to be stored for later use in moderator functions or other user-written subprograms. The values of these user-defined task characteristics may be manipulated through the use of user support subprograms GETTC and PUTTC. The definitions of these subroutines are as follows:

Subroutine	Function
GETTC(NTASK,NCHAR, VALUE)	Retrieves VALUE as the value of user-defined task characteristic NCHAR of task NTASK.
PUTTC(NTASK,NCHAR, VALUE)	Sets user-defined task characteristic NCHAR of task NTASK to VALUE.

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, re-read the section.

1. The SAINT user may define characteristics of tasks other than those required by SAINT by specifying the values of user-defined task *characteristics*.

2. Subroutine GETTC retrieves values of user-defined task characteristics.

3. Subroutine PUTTC resets values of user-defined task characteristics.

SECTION XX

USER-GENERATED INPUT AND OUTPUT

In the previous sections, we have been shown that we can write our own subprograms (USERF, MODRF, PRIOR, etc.) to interact with the SAINT simulation. In writing these routines, we may define our own variables to serve as coefficients for the state variable equations or conditions for the user and moderator functions we write; or we may use resource attributes, system attributes, switch values, etc., for the same purpose. In either case, it might prove necessary or beneficial to generate our own input and output reports relating to these variables.

SAINT includes a number of subprograms that allow us to manipulate input and output information. Subroutine UINPT can be coded to read user-defined input data. Subroutines UCLCT, UTMST, UTMSA, UHIST, and UPLOT are used to generate statistics. Subroutines CLROB, CLRTP, CLRHI, CLRPT, and CLEAR are used to initialize statistical information storage arrays. Subroutines ENDIT AND UOTPT are supplied for the generation of user-required output information.

Desired Statistics Collection for the GCV Model

Let us return to the GCV system that we finalized in Section XVII. Suppose that we would like to perform a series of iterations and obtain statistics on the following:

1. The utilization of the operator (averaged over all iterations).

2. The average flight deviation (in nautical miles) of each GCV during each iteration.

3. A plot of the deviation (in nautical miles) of each GCV for each iteration.

4. The average deviation (in nautical miles) of each GCV from the target when it is at x-coordinate 500,000 feet (averaged over all iterations).

The first statistic listed, the utilization of the operator, is provided automatically by SAINT. For each resource included in a model, SAINT automatically provides statistics on the amount of time that the resource is busy and idle averaged over all iterations. The other statistics listed above, however, involve calculations using state variable values as the iteration progresses. Since these statistics are not automatically produced by SAINT, and cannot be requested using task, resource, or state variable statistics directly, we will use some of the special output routines available in SAINT to generate these statistics.

User-Generated Input

If we require special input data for moderator functions or other user-written subprograms, we employ subroutine UINPT. This subroute is called automatically by SAINT following the processing of all SAINT data input cards. Thus, all of the data to be read by this subroutine must follow the SAINT data cards. The only restriction on the preparation of subroutine UINPT is that it must be written in FORTRAN or a compatible language. All standard language conventions must be observed.

If input information is to be stored for later use, then we are required to develop this storage mechanism. All arrays must be dimensioned. In addition, only labeled COMMON blocks are allowed. We must be certain not to use a SAINT COMMON variable as storage for our input data. If we have no special data to input, this subroutine need not be written.

Collection of Statistics Based on Observation

To collect sample data for variables based on observation, we use subroutine UCLCT. The statement CALL UCLCT (XX,ICODE) causes SAINT to record the value XX as an observation of the variable with code ICODE. For example, in our GCV system, we are to record the average deviation (in nautical miles) of each GCV from the target when it is at an x-coordinate of 500,000 feet.

Figure 56 (a duplication of Figure 55) illustrates our most recent GCV model. In that model, task 5 is signaled when the first GCV is at its 500,000 foot x-coordinate. At this time, SS(3) is the deviation of the GCV in feet. If we define statistic code 1 for variables based on observation (ICODE=1) as the deviation of GCV 1 at this point, we can compute this deviation and collect the value when task 5 is started. To accomplish this collection, we request a moderator function to be active at task 5 which contains the following statements:

```
XX = SS(3)/6080. [conversion of SS(3) to nautical miles]

CALL UCLCT(XX,1)
```

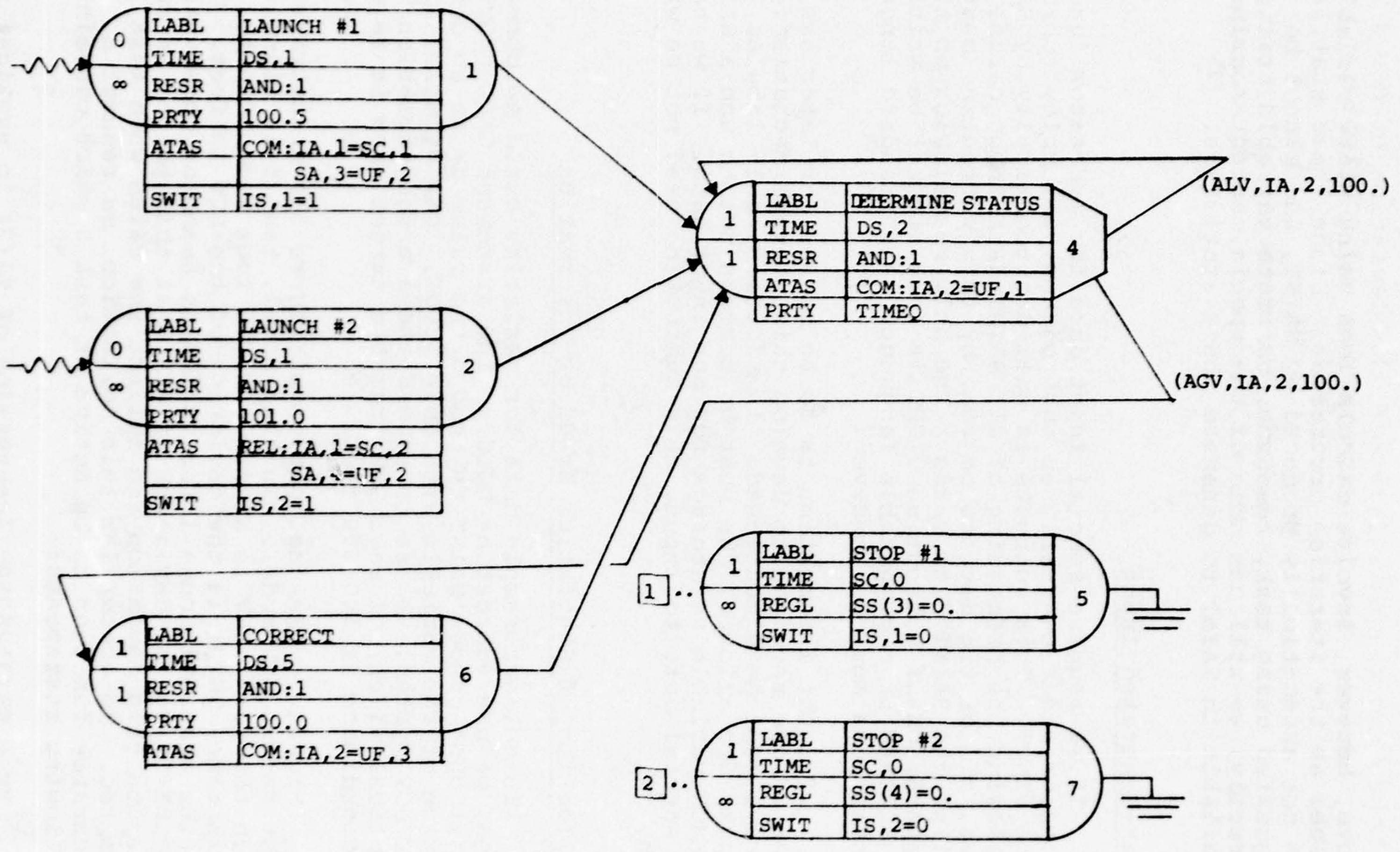


Figure 56. SAINT Model Illustrating Regulation.

In this manner, we will record the deviation of GCV 1 in nautical miles when it achieves its 500,000 foot x-coordinate.

We use similar statements (referencing SS(4) and setting ICODE to 2) within a moderator function called at task 7 to record the deviation of GCV 2.

Statistics based on observation may be collected for any variable in a SAINT simulation. However, a unique ICODE must be defined for each unique statistic that we collect.

Collection of Statistics for Time-Persistent Variables

A time-persistent variable is, as one might expect, a variable whose value persists over time. For example, consider an operator performing tasks in a system. When he is busy, the operator will be busy for a certain period of time (his busy status persists over time). Similarly, when he becomes idle, he will be idle for a period of time.

Subroutine UTMST

We can use subroutine UTMST to collect sample data for time-persistent variables that we can assume do not change value during the time interval between statistical collection points. The statement CALL UTMST(XX,T,ICODE) causes SAINT to record a status change for the time-persistent variable with code ICODE. The status change is defined as the time at which the value of the variable is changed, T, and the new value of the variable, XX.

Remember that the value of XX used as an argument for UTMST should be the value of the variable after it is changed by a simulation event. For multiple iterations, the resetting of the initial value of the variable of interest is our responsibility (refer to subroutine ENDIT discussion later in this section). The variable USTPV(ICODE,6) contains the last (or initial) value of the variable.

To illustrate the use of subroutine UTMST, consider Figure 57, assuming that we have a simulation in which there are two resources and that we are interested in the number of the two resources of the system that are busy at any point in time. Initially, both resources are busy. At time 25, resource 1 finishes the task it is performing and is idle for the remainder of the simulation. At time 50, resource 2 finishes the task it is performing and the simulation ends. To record the number of resources busy over time, we use the following calls to subroutine UTMST with statistic code

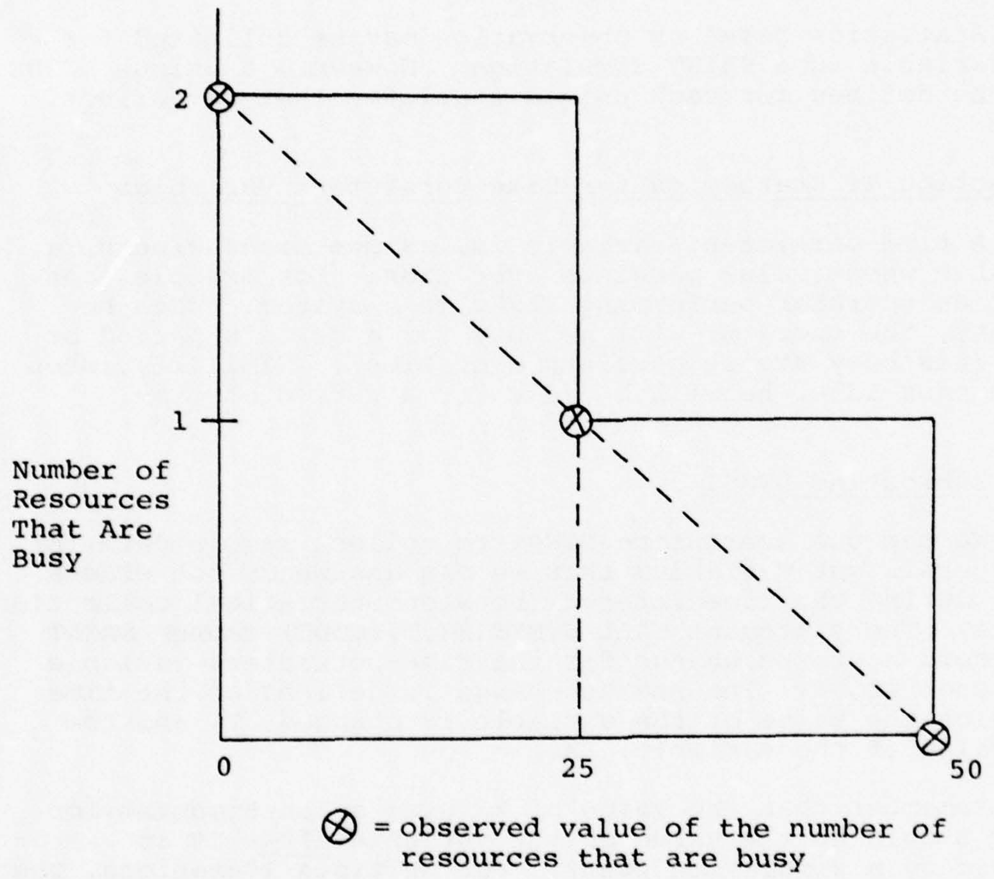


Figure 57. The Use of Subroutines UTMSA and UTMST.

ICODE = 1 (the call at time 0 is performed automatically by SAINT):

Time	Call
0	CALL UTMST(2.,0.,1)
25	CALL UTMST(1.,25.,1)
50	CALL UTMST(0.,50.,1)

When requested to compute the statistic based on the above calls, SAINT would compute the area under the curve in Figure 57 given by the solid line (75) and divide by the total time (50) to calculate the average number of resources that were busy at any point in time during the simulation (1.5).

Subroutine UTMSA

In addition to the time-persistent variables discussed above, we may want to collect sample data for time-persistent variables that may change value during the time interval between statistical collection points. Subroutine UTMSA is used for this purpose. It collects information identical to the information collected by subroutine UTMST, except that an average value during the past interval is assumed as an approximation to the value during the interval.

To illustrate the use of subroutine UTMSA, again consider Figure 57. The variable of interest remains the number of the two resources of the system that are busy. However, assume that the resources are busy and idle many times throughout the time interval 0 to 50, and that the number of resources busy is sampled at times 0, 25, and 50 as follows:

Time	Call
0	CALL UTMSA(2.,0.,1)
25	CALL UTMSA(1.,25.,1)
50	CALL UTMSA(0.,50.,1)

Here, UTMSA is called to approximate the average number busy from selected samples. When SAINT reports on this statistic, it would compute the area under the curve given by the dashed line in Figure 57 (50) and divide by the total time (50) to calculate the average number of resources that were busy at any point during the simulation (1.0).

Statistics on Time-Persistent Variables for the GCV System

Earlier, we decided to collect statistics on the average flight deviation (in nautical miles) of each GCV during each iteration. Ideally, we would like to continuously monitor the deviation of each GCV and integrate the curve of its deviation over time in order to determine the average deviation. However, let us treat the deviation of each GCV as a time-persistent variable and use subroutine UTMSA to approximate the statistic.

In Figure 56, the operator continuously monitors the status of the two GCVs (task 4). At the end of the monitor task, user function 1 is called to determine if the deviation of the GCV is greater than 100 feet. Within user function 1, we can execute the following statements (we define statistic code 1 as the deviation of GCV 1 and statistic code 2 as the deviation of GCV 2) in order to collect statistics on GCV deviation:

```
XX=SS(I+2)/6080.

T=TNOW

CALL UTMSA(XX,T,ICODE)
```

Note that in the above statements, we used the SAINT COMMON variable TNOW. At the time the user function is called, this variable is equal to the current value of simulated time. By using the above procedure, the statistic for the deviation of each GCV will be based on the average value of the deviation between the times that the GCV is monitored by the operator.

Collection of Histogram Information

In addition to collecting statistics on the values of variables based on observation, we may also produce histograms of those values that we record. To collect histogram information, we employ subroutine UHIST. The use of this subroutine is identical to the use of subroutine UCLCT. For example, assume that we wish to generate a histogram of the values of the deviation of each GCV at the times its x-coordinate is 500,000 feet. To do so, we add the following statement to the moderator functions associated with tasks 5 and 7:

```
CALL UHIST(XX,ICODE).
```

By employing subroutine UHIST, we can produce a histogram of the values of the devaition of the GCVs as well as collecting statistics on them using subroutine UCLCT.

Collection of Plot Information

With SAINT, we have capability of plotting values of variables versus an independent variable. We can collect values of up to 10 dependent variables for each independent variable, and we may specify multiple plots. Subroutine UPLOT is used for this purpose. The statement CALL UPLOT(X,T,IPLOT) causes the values of the variables contained in the array X to be plotted versus the independent variable T on plot IPLOT.

For our example, assume that we wish to plot the deviation of the GCVs over time. Thus, in user function 1, whenever we monitor the status of the GCV, let us plot out the deviations of both GCVs at this point in time. To request this plot, we define variable 1 of plot 1 to be the deviation of GCV 1 (in nautical miles). We then include the following statements in user function 1 (the statement DIMENSION X(2) must also be included).

```
X(1)=SS(3)/6080.

X(2)=SS(4)/6080.

T=TNOW

CALL UPLOT(X,T,1).
```

The only restriction on the plotting capability in SAINT is that the independent variable, T, must be monotonically non-increasing or non-decreasing.

Reporting Statistics Collected for an Iteration

In the preceding paragraphs, we have seen how to collect statistics on variables that we have defined in user-written subprograms. However, since SAINT does not automatically output the information we desire, we must tell the program when to do so.

We inform SAINT to output the information that we desire by specifying special values of ICODE and IPLOT in our calls to UCLCT, UHIST, UTMST, and UPLOT. If the statistic code is preceded by a minus sign, then SAINT will output a report on the statistic defined by that code.

For example, the statement CALL UCLCT (XX,-1) will cause SAINT to produce an output report for statistic 1 for variables based on observation. This report consists of the mean, standard deviation, standard deviation of the mean (assuming independent observations), coefficient of variation, minimum, maximum, and number of observations of the sampled variable.

Similarly, the statement CALL UHIST(XX,-2) will cause SAINT to produce a summary of the number of times the variable defined by statistic code 2 is within prescribed cell limits (as defined on input). It calculates the observed, relative, and cumulative frequencies for each cell. It also provides a plotted histogram displaying the relative and cumulative frequency of observations.

For variables collected using both subroutines UTMSA and UTMST, a call to subroutine UTMST is used to obtain output reports. Whether the time-persistent variable that we define is collected using UTMSA or UTMST, the statement CALL UTMST(XX,T,-3) will cause information on the time-persistent statistic defined by code 3 to be produced at time T. The information produced includes the mean, standard deviation, minimum, and maximum of the sampled variable. Note that because UTMST and UTMSA are highly related, the codes for time-persistent variable statistics must be unique, i.e., we cannot use a call to UTMSA with a code of 1 and also use a call to UTMST with a code of 1.

To obtain plotted output, the statement CALL UPLOT(X,T,-1) can be used. When this statement is executed, SAINT automatically provides a plot of the data collected for plot 1.

It should be noted that in the above calls to these routines, the value XX or the array X is passed as an argument. However, when the statistic code is preceded by a minus sign, the values XX and X are not used.

If we wish to output all histograms, all statistics based on observation, etc., we call the appropriate routine with ICODE = 0. For all the user-generated statistical collection routines, a call with the argument ICODE = 0 will produce output reports for each statistic that we have defined. For example, the statement CALL UCLCT(XX,0) produces the statistical output for all user-generated statistics based on observation.

User-Generated Statistics for the GCV System

We have seen how the statistical collection routines can be used for our GCV system. The following discussion and figures summarize these procedures and show the specific coding of the SAINT model of the GCV system to obtain the aforementioned statistics. The model of the GCV system is shown in Figure 58, function USERF in Figure 59, and function MODRF in Figure 60.

Remember that whenever it is necessary to use a SAINT COMMON variable in a user-written subprogram, the SAINT labeled COMMON block that contains the variable must be included in the subprogram.

Function USERF

The FORTRAN code for function USERF is shown in Figure 59. This code is unchanged from the previous user function (Figure 54) except for the inclusion of user-generated statistical requests. User function 1 is called from task 4 to compute the deviation of the GCV on which the operator is currently working. Within user function 1, we want to collect time-persistent statistics (using subroutine UTMSA) for the deviation of the GCV on which the operator is working. We have defined time-persistent statistic 1 as the deviation of GCV 1 and time-persistent statistic 2 as the deviation of GCV 2. Thus, the statistic code for the call to UTMSA is equal to the GCV number currently being monitored. To collect statistics on the deviation, (SS(3) for GCV 1 and SS(4) for GCV 2), we set XX to the appropriate SS(•) value divided by the conversion factor (feet to nautical miles), set T to the current time TNOW, set ICODE to the GCV number, and call subroutine UTMSA.

In addition to collecting time-persistent statistics on the deviation, we also wish to plot the deviation of both GCVs. Thus, we set the array elements X(1) and X(2) to the deviations of GCV 1 (SS(3)/6080.) and GCV 2 (SS(4)/6080.), respectively). We then call subroutine UPLOT with the array elements, the current time, and the plot number (which we have defined as 1).

Subroutine MODRF

The FORTRAN code for function USERF collects statistics and plot information on the deviations of the GCVs over time. To collect the observation statistics on the deviations of the GCVs when they pass the X-coordinate of 500,000 feet, we employ two moderator functions. Moderator function 1 is called from task 5 to record the status of GCV 1, while moderator function 2 is accessed at task 7 to record the deviation status of GCV 2. Within each moderator function, as shown in Figure 60, we set XX to the value of the SS(•) variable representing the current deviation of the

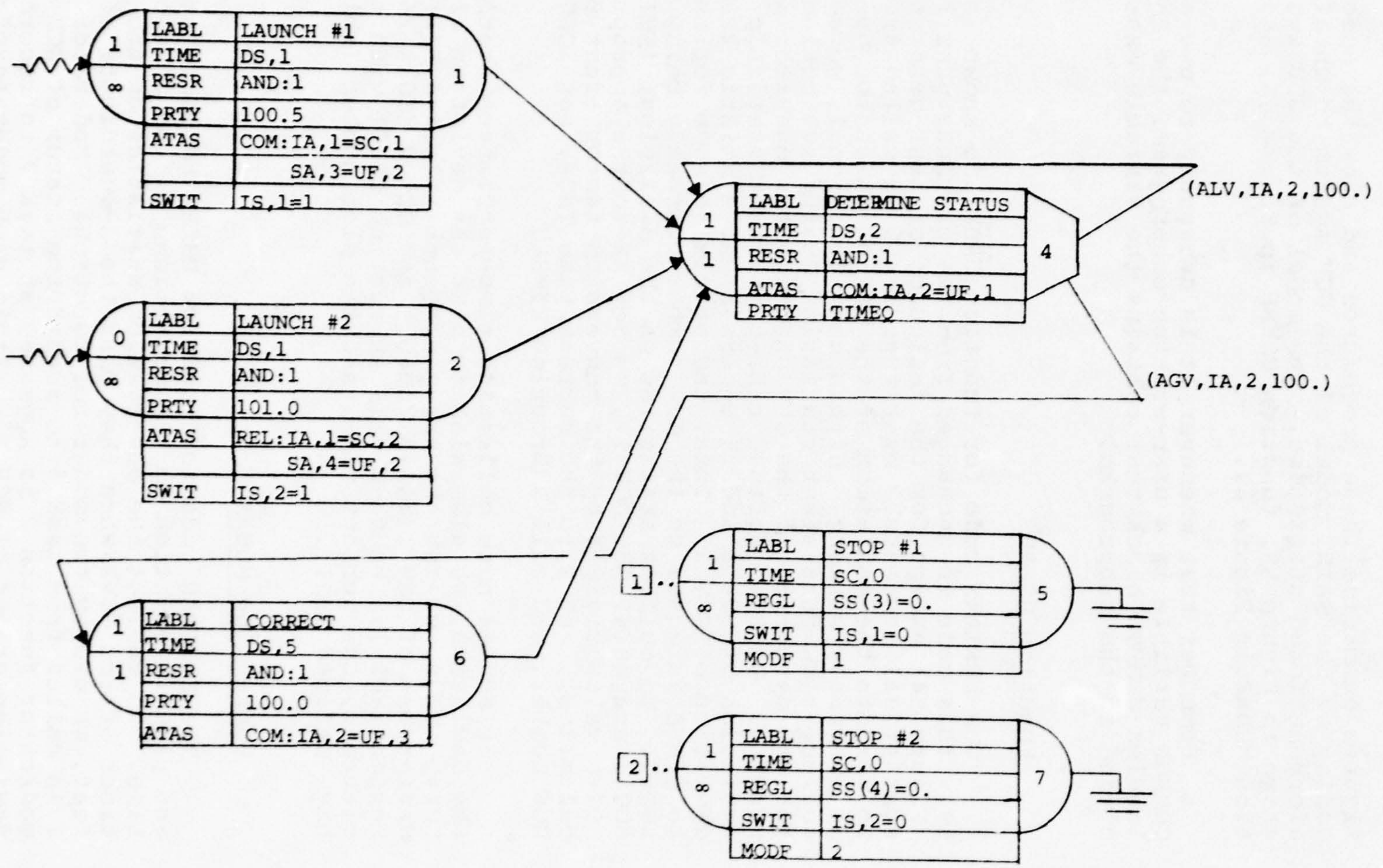


Figure 58. SAINT Model Illustrating User-Generated Statistics.

```
      FUNCTION USERF(IP)
C
      COMMON /COM06/ TNOW,TTNCX,MFAD,SEED,ISEED,NCRDR,NPRNT,NPUNCH,
     *               NRNIT,MNDC,NDC,NDTN,NNTC
      COMMON /COM17/ SS(100),SSL(100),DD(100),DDL(100),LLRES(100,2)
      DIMENSION X(10)
C
      GO TO (10,20,30),IP
C
C**** CODE FOR USER FUNCTION 1
C
   10 CALL GETIA(1,VALUE)
      IGCV=VALUE
      INDX=IGCV+2
      USERF=SS(INDX)
      ICODE=IGCV
      XX=SS(INDX)/6080.
      T=TNOW
      CALL UTMSA(XX,T,ICODE)
      X(1)=SS(3)/6080.
      X(2)=SS(4)/6080.
      CALL GPLOT(X,T,1)
      RETURN
C
C**** CODE FOR USER FUNCTION 2
C
   20 CALL GETIA(1,VALUE)
      IGCV=VALUE
      INDX=IGCV+2
      CALL GETSA(INDX,VALUE)
      USERF=VALUE+RNORM(4)
      CALL CLRTP(IGCV)
      RETURN
C
C**** CODE FOR USER FUNCTION 3
C
   30 CALL GETIA(1,VALUE)
      IGCV=VALUE
      XX=SS(IGCV)
      YY=SS(IGCV+2)
      ALPHA=ASIN(Y/(500000.-X))
      HEAD=ALPHA+RNORM(4)
      CALL PUTSA(IGCV+2,HEAD)
      RETURN
      END
```

Figure 59. Function USERF for SAINT Model of Figure 58.

```
      SUBROUTINE MODRF(MODFN,NTASK)
C
      COMMON /COM06/ TNOW,TTNEX,MFAD,SEED,ISEED,NCRDR,NPRNT,NPUNCH,
     *               NRNIT,MNDC,NDC,NDTN,NNTC
      COMMON /COM17/ SS(100),SSL(100),DD(100),DDL(100),LLRES(100,2)
C
      GO TO (10,20), MODFN
C
C**** CODE FOR MODERATOR FUNCTION 1
C
   10 XX=SS(3)/6080.
      CALL UCLCT(XX,1)
      CALL UHIST(XX,1)
      T=TNOW
      CALL UTMST(XX,T,-1)
      RETURN
C
C**** CODE FOR MODERATOR FUNCTION 2
C
   20 XX=SS(4)/6080.
      CALL UCLCT(XX,2)
      CALL UHIST(XX,2)
      T=TNOW
      CALL UTMST(XX,T,-2)
      RETURN
      END
```

Figure 60. Subroutine MODRF for SAINT Model of Figure 58.

appropriate GCV divided by 6080. We then call subroutines UCLCT and UHIST to record the values (statistic 1 is used for GCV 1 and statistic 2 is used for GCV 2).

The deviation statistics are collected by subroutine UTMSA from time 0 until the GCV has passed the 500,000 foot x-coordinate. Thus, the output request for the statistics collected by subroutine UTMSA is made at the time that the GCV passes the 500,000 foot x-coordinate. Since we know that this occurs when moderator function 1 is called for GCV 1 and when moderator function 2 is called for GCV 2, we request the output for the time-persistent statistics on the deviations of these GCVs within moderator functions 1 and 2. Remember that the request for output for statistics collected by subroutine UTMSA is made through a call to subroutine UTMST. The statement CALL UTMST(XX,T,-1) in moderator function 1, where XX is not defined for this call, T is the current time, and -1 is the negative of the statistic that we desire, causes SAINT to produce the output report for the time-persistent statistic on the deviation of GCV 1. Likewise, the call to UTMST(XX,T,-2) in moderator function 2 causes SAINT to report on the deviation statistic for GCV 2.

Subroutine ENDIT

We have collected all the desired statistics and have reported on the time-persistent deviation statistics. However, we have not yet prepared the plot of GCV deviation for each iteration. Subroutine ENDIT is the user-written subprogram employed to request the plotted output. It is automatically called by SAINT at the end of each iteration with one argument, the number of the iteration that was just completed. This subroutine is particularly useful for causing the calculation and reporting of user-generated statistics and the reinitialization of the statistical storage arrays. In addition, special iteration summary reports can be prepared.

The FORTRAN code for subroutine ENDIT is shown in Figure 61. In order to cause SAINT to prepare the plot, we call subroutine UPLOT with an unused but dimensioned X array, the current time, and IPLOT = -1. When this call is made, a plot of the deviations of both GCVs will be printed.

Initializing Statistics

It was stated above that subroutine ENDIT is also useful for the reinitialization of statistical storage arrays. We may perform the reinitialization of statistical

```
      SUBROUTINE ENDIT(ITER)
C
      COMMON /COM06/ TNOW,TTNEX,MFAD,SEED,ISEED,NCRDR,NPRNT,NPUNCH,
     *               NRNIT,MNDC,NDC,NDTN,NNTC
      COMMON /COM23/ LLUGC(20,2),USOBV(20,5),LLUGT(20,2),TTCLR(20),
     *               USTPV(20,6),LLUGH(20,2),NNCEL(20),HHLOW(20),
     *               HHWID(20),JJCEL(540)
      DIMENSION X(10)
C
C**** PROCESS END OF ITERATION STATISTICS
C
      T=TNOW
      CALL UPLOT(X,T,-1)
C
C**** REINITIALIZE FOR NEXT ITERATION
C
      USTPV(1,6)=0.
      USTPV(2,6)=0.
      RETURN
      END
```

Figure 61. Subroutine ENDIT for SAINT Model of Figure 58.

storage arrays by using any of several SAINT subroutines developed for this purpose. We may call subroutine CLEAR, which initializes the user-generated statistical storage arrays for variables based on observation, time-persistent variables, histograms, and plots. This subroutine is automatically called by SAINT following the reading of all data cards associated with user-generated statistics to prepare for statistical collection. However, we may employ CLEAR at any time during the simulation by a direct call.

In addition to subroutine CLEAR, which initializes all statistical storage arrays, we may selectively clear statistical storage arrays through the use of one of four other subroutines. Subroutine CLROB initializes the statistical storage arrays associated with the collection of user-generated statistics for variables based on observation (collected using subroutine UCLCT); subroutine CLRTP initializes the statistical storage arrays associated with the collection of user-generated statistics for time-persistent variables (collected using subroutines UTMST and UTMSA); subroutine CLRHI initializes the statistical storage arrays associated with the preparation of user-generated histograms (collected using subroutine UHIST); and subroutine CLRPT initializes the storage arrays and peripheral storage units associated with the preparation of user-generated plots (collected using subroutine UPLOT).

All of the above routines have one argument, IND. This argument allows us to selectively clear the storage for any particular statistic or for all statistics. If IND is positive, then the clearing is done only for statistic number IND. If IND is 0, then the statistical storage arrays of all statistics are cleared.

Initialization Performed by SAINT

The statistical storage for user-generated statistics based on observation and user-generated histograms is initialized automatically by SAINT only following the reading of user-generated statistics data cards. After that time, we are totally responsible for any initialization required, including the end of an iteration and following computation and reporting.

For time-persistent variables, SAINT automatically performs the initialization at the beginning of each iteration. However, the SAINT variable USTPV(ICODE,6), the initial value of the time-persistent variable, must be reset by the user for each iteration. Otherwise, it will retain its most recent value.

For user-generated plots, the initialization of storage for all plots is performed automatically by SAINT following the reading of data cards describing the user-generated plots and following computation and reporting. Otherwise, we are totally responsible for initialization of plot storage including the end of an iteration for which we do not request plotted output.

Initialization of the GCV Model

In our model, we desire the plotting information to be reinitialized following each iteration. However, since we request the plot at the end of each iteration, this reinitialization is automatically performed by SAINT.

We desire the statistics for time-persistent variables to also be reinitialized following each iteration. As this initialization is automatically performed by SAINT, we do not have to call subroutine CLRTP from subroutine ENDIT.

Also, since we desire the statistics based on observation and the histograms to be collected over all iterations, we need not specify any clearing for these statistics in subroutine ENDIT.

However, since we are collecting time-persistent variable statistics, we are required to reset the initial value of each statistic. In subroutine ENDIT, we set the initial values of the deviation of both GCVs for statistical collection purposes to 0. Thus, we set USTPV(1,6) and USTPV(2,6), for GCVs 1 and 2, respectively, to 0.

At this point, we should recognize that we are collecting deviation statistics for the GCVs from time 0 until the time they reach x-coordinate 500,000. It is probable that we would rather record statistics on the deviations of the GCVs from the time that they are launched until they reach their final x-coordinate. We would like to reinitialize the statistical storage arrays for the time-persistent variables when these GCVs are launched. If we refer to Figure 58, we note that when either GCV 1 or GCV 2 is launched, user function 2 is called. Thus, in the FORTRAN code for user function 2 shown in Figure 59, we want to clear the statistical storage arrays for the time-persistent variable of the appropriate GCV. Since the statistic code and the GCV number are the same, we simply call subroutine CLRTP with the GCV number as the argument. This accomplishes the required reinitialization.

Reporting Statistics Collected Over All Iterations

The last statistics of interest that we have yet to output are the statistics based on observation and the histograms of the deviation of the GCVs when they reach their final x-coordinate. To output these statistics after all iterations have been completed, we employ subroutine UOTPT. Subroutine UOTPT is automatically called by SAINT at the end of the entire set of iterations that are executed. This subroutine is most helpful for the preparation of special simulation reports which require information collected over a series of iterations. Thus, in the FORTRAN code for subroutine UOTPT shown in Figure 62, we request all the information for user-generated statistics based on observation and histograms. We request this information by calling subroutines UCLCT and UHIST with 0 arguments, causing SAINT to print reports on all user-generated statistics based on observation and histograms that we have defined.

Summary

The following SAINT modeling concepts were discussed in this section. If you do not understand these concepts, re-read this section.

1. Subroutine UCLCT(XX,ICODE) is used for collection and reporting of user-generated statistics for variables based on observation.

2. Subroutine UTMST(XX,ICODE) is used for collection and reporting of user-generated statistics for time-persistent variables.

3. Subroutine UTMSA(XX,ICODE) is used for collection of user-generated statistics for time-persistent variables.

4. Subroutine UHIST(XX,ICODE) is used for collection and reporting for user-generated histograms.

5. Subroutine UPLOT(X,T,IPLOT) is used for collection and reporting of user-generated plots.

6. Subroutine CLEAR is used for reinitialization of all user-generated statistics and plots.

7. Subroutine CLROB(IND) is used for reinitialization of user-generated statistics for variables based on observation.

```
      SUBROUTINE UOTPT
C
C**** PROCESS END OF SIMULATION STATISTICS
C     XX=0.
      CALL UCLCT(XX,0)
      CALL UHIST(XX,0)
      RETURN
      END
```

Figure 62. Subroutine UOTPT for SAINT Model of Figure 58.

8. Subroutine CLRTP(IND) is used for reinitialization of user-generated statistics for time-persistent variables.

9. The SAINT variable USTPV(ICODE,6) must be reset by the user when user-generated statistics for time-persistent variables are reinitialized.

10. Subroutine CLRHI(IND) is used for reinitialization of user-generated histograms.

11. Subroutine CLRPT(IND) is used for reinitialization of user-generated plots.

12. Subroutine ENDIT is used to generate reports of statistical quantities for an iteration and to reinitialize user-generated statistics for the next iteration.

13. Subroutine UOTPT is used to generate reports of statistical quantities for a set of iterations.

SECTION XXI

TASK MODIFICATION AND TASK CLEARING

This section and the two following present additional capabilities that offer us alternatives for modeling in SAINT. These capabilities are task modification, distribution set modification, task clearing, resource clearing, and the specification of a task completion precedence. In order to provide a more general framework for the discussion of these concepts, let us expand our model of the GCV system so that the tasks in the network relate to specific GCVs.

An Expanded GCV Model

Our new GCV model is presented in Figure 63. Operationally, it is equivalent to the model in Figure 58. However, the tasks representing monitoring and correcting GCV status have each been divided into two tasks. Tasks 1 and 2 still represent the launch of GCVs 1 and 2, respectively. However, task 3 represents the operator monitoring GCV 1, while task 4 represents the operator monitoring GCV 2. Similarly, task 5 represents the operator correcting the status of GCV 1, while task 6 represents the operator correcting the status of GCV 2. Tasks 7 and 9 are sink tasks and are equivalent to tasks 5 and 7, respectively, of the model in Figure 58. During execution of the model, the operator will launch GCV 2, then launch GCV 1, then monitor and correct (if necessary) GCV 2, then monitor and correct (if necessary) GCV 1, then monitor and correct (if necessary) GVC 2, etc., until both GCVs have passed the x-coorindate of 500,000 feet.

When GCV 1 passes its limiting x-coordinate, the operator continues to monitor this GCV but never corrects its flight. We have modeled this by setting the state variables and switch values such that the deviation status of the GCV will always be 0 beyond that time. However, what if we do not even want the operator to monitor the status of the GCV once it has passed the x-coordinate value of 500,000? In that case, when task 7 is signaled (specifying that GCV 1 has passed its 500,000 x-coordinate), we would like to prevent him from any further activity on GCV 1. This can be accomplished through the use of task clearing and task modification.

Description of Task Modification

Task modification involves a substitution of the characteristics and output side of one task for those of another

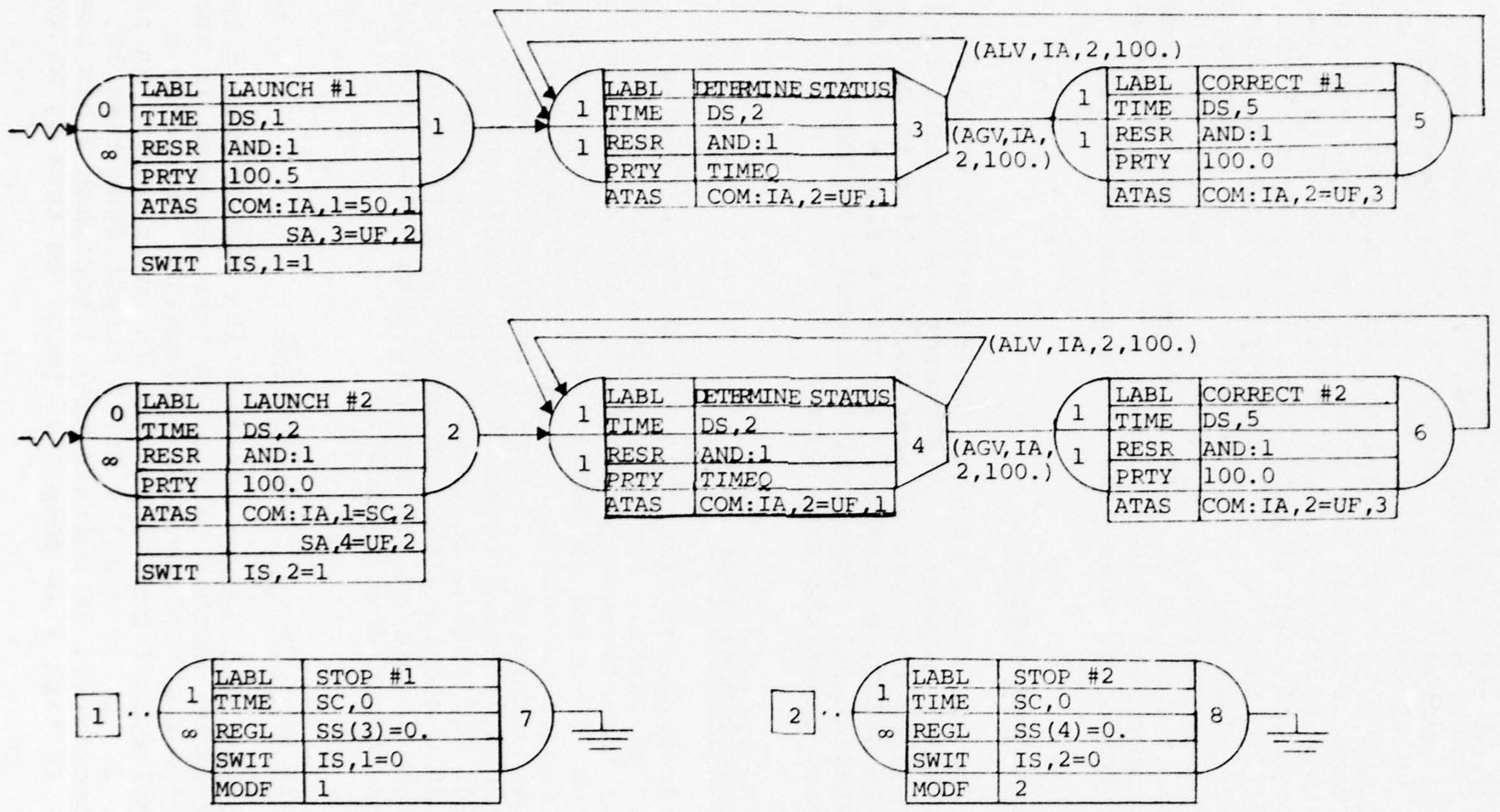


Figure 63. Expanded SAINT Model of the GCV System.

task upon the completion of a third task. The SAINT representation of task modification is presented in Figure 64. A dashed line is drawn between the original task in the network and the task to be substituted for this task upon completion of another task. In the triangle appearing at the side of this dashed line is the number of the task whose completion causes the modification. In Figure 64, the completion of task 20 causes the substitution of the description and output portions of task 30 for those of task 10. It should be noted that if task 10 is in progress at the time the modification is made, then branching will occur from task 30. However, if the task is not in progress at the time of the completion of task 20, the entire descriptive section of task 30 is used to determine the resources required, performance time required, and any other operations to be performed when task 10 is again scheduled.

Types of Task Modification

There are two types of task modifications which may appear in a SAINT network: interchange and multiple replacement. We specify the type of task modification to be used on input. In SAINT, we may only specify a single type for an entire simulation. Thus, all task modifications within a network must be of the same type.

Interchange Modification

An interchange modification allows for reflexive substitution of tasks. Figure 65 illustrates a situation requiring an interchange modification. In the example, an input to task 10 causes an output from task 10 and an input to task 20 causes an output from task 20 if tasks 1 and 2 have not been completed. If task 1 is completed prior to task 2, an input to task 10 or 20 would result in an output from task 20. If task 2 is then completed, an input to task 10 or 20 would result in an output from task 10. If task 1 is then completed, an input to task 10 or 20 would result in an output from task 20.

Multiple Replacement Modification

A multiple replacement modification occurs when a task that has replaced another task is in turn replaced in the network. Figure 66 illustrates a situation requiring a multiple replacement modification. In the example, an input to task 10, 20, or 30 results in an output from task 10, 20, or 30, respectively, if neither task 1 nor task 2 has been completed. If task 1 is completed prior to task 2, an input

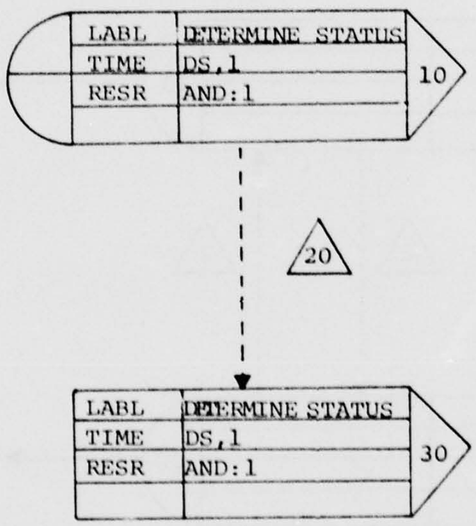


Figure 64. SAINT Representation of Task Modifications.

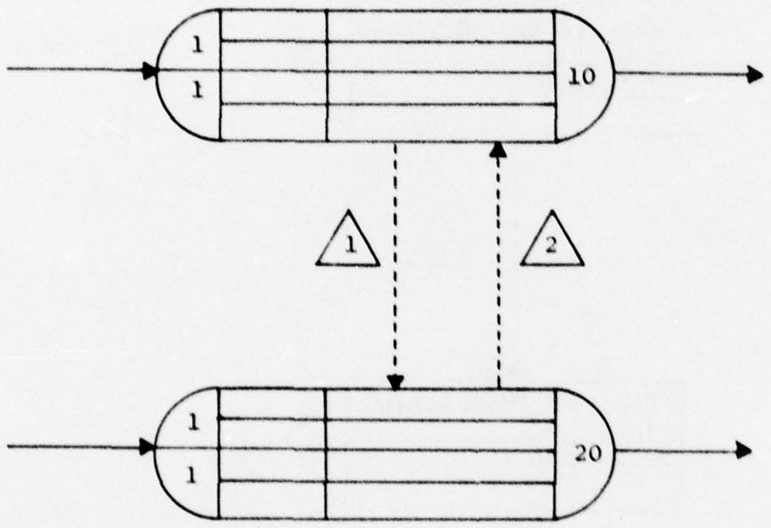


Figure 65. An Example of Interchange Modifications.

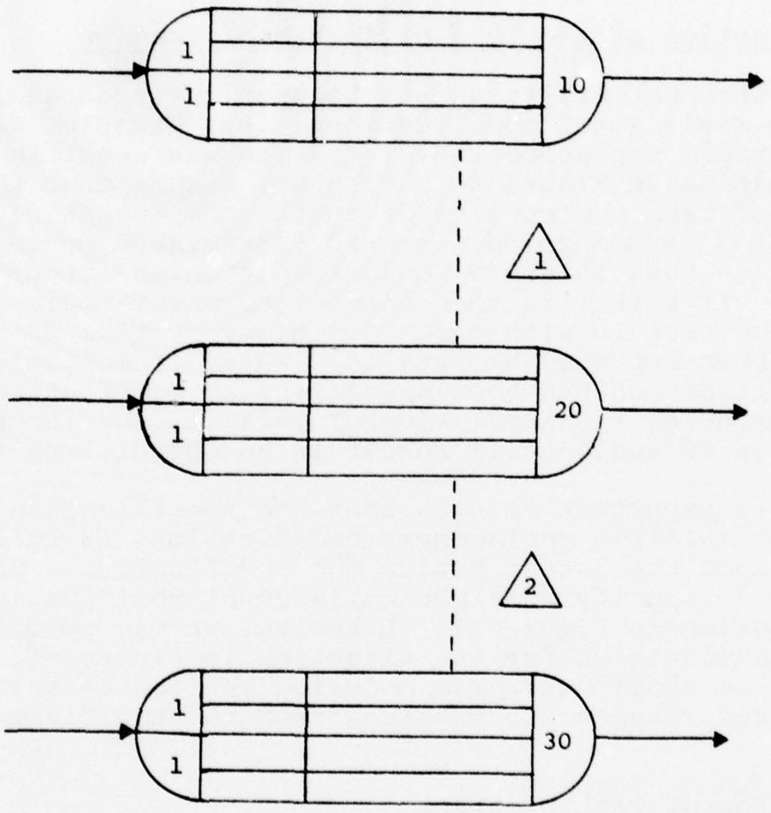


Figure 66. An Example of Multiple Replacement Modifications.

to task 10 or 20 results in an output from task 20 and an input to task 30 yields an output from task 30. If task 2 is then completed, an input to task 10, 20, or 30 results in an output from task 30. If task 2 is completed prior to task 1, an input to task 10 yields an output from task 10 and an input to task 20 or 30 results in an output from task 30. If task 1 is then completed, an input to any of the three tasks would result in an output from task 30.

Selection of the Type of Task Modification

An incompatibility exists between interchange and multiple replacement modifications. For example, assume that multiple replacement modification was specified for the situation in Figure 65. Further, assume that the completion of task 2 occurs prior to the completion of task 1. When task 2 is completed, task 20 is replaced by task 10. An input to task 10 or 20 would result in an output from task 10. If task 1 is then completed, SAINT would attempt to replace task 10 with task 20. However, task 20 has already been replaced by task 10. Thus, if multiple replacement modification had been specified, no modification would occur following the completion of task 1. Any inputs to task 10 or 20 would still result in an output from task 10.

It is important to note that the specification of interchange or multiple replacement modifications is solely dependent upon the intent behind the modification. SAINT does allow us to specify multiple replacement modification for the situation in Figure 65. Likewise, we may specify interchange modification for the situation in Figure 66. Because of this, we should exercise modeling caution to insure that the desired results are obtained from the modification.

Description of Task Clearing

Task clearing involves the interruption (clearing) of a task in progress when another task is completed. If a clearing operation is performed, SAINT allows a task to be signaled. The number of predecessor requirements for the signaled task will be reduced by 1. Clearing information is included as part of the description of the task whose completion causes the clearing. The task description code for task clearing is TCLR. In the right-hand side of the TCLR row appears the letter C followed by the resource to be cleared and, if appropriate, the letter S followed by the task to be signaled. For example, in Figure 67, the completion of task 10 causes the clearing of task 20 and the signaling of task 30. In addition, since we signal a task, SAINT provides a mechanism for specifying the signaling

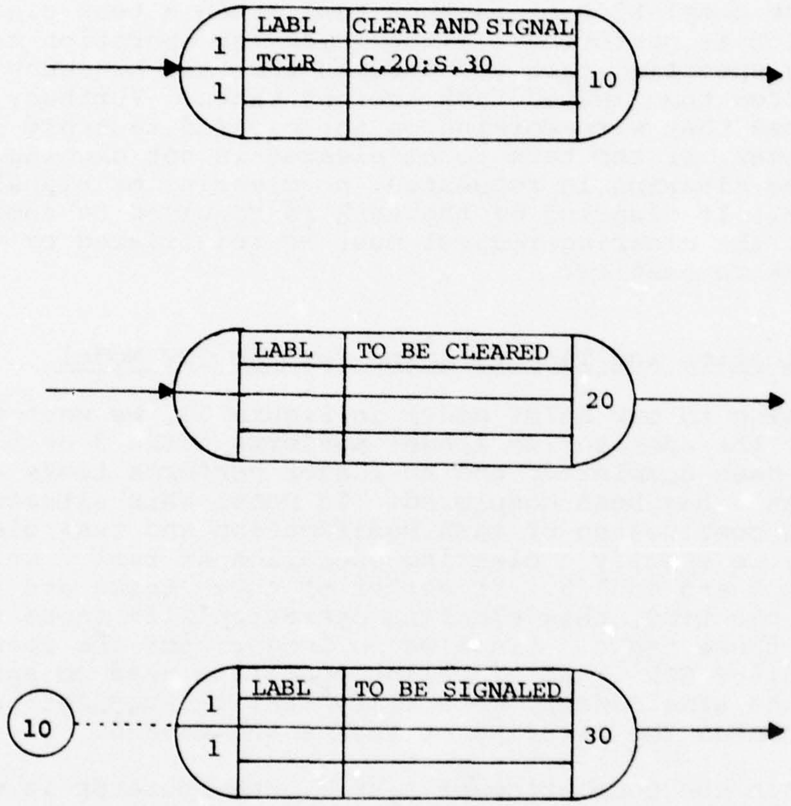


Figure 67. SAINT Representation of Task Clearing.

operation at the task to be signaled. A circle is connected to the input side of the task to be signaled by a dotted line. Inside the circle, we place the number of the task that caused the signaling operation. Thus, in Figure 67, the circle specifies that task 3 is signaled whenever a clearing operation is performed upon the completion of task 10.

In summary, if it is desired to halt a task in progress based on the completion of another task, then a task clearing operation is performed. A task clearing operation halts the ongoing specified task and assumes that the branches emanating from the cleared task are not taken. Further, all resources that were working on the cleared task are set idle. However, if the task to be cleared is not ongoing at the time the clearing is requested, no clearing or signaling takes place. If clearing of the task is required at some later time, the clearing request must be reinitiated by another task completion.

Task Modification and Task Clearing for the GCV Model

Returning to our SAINT model in Figure 63, we want to insure that the operator no longer performs tasks 3 or 5 if task 7 has been completed; and no longer performs tasks 4 or 6 if task 8 has been completed. To model this situation, we employ a combination of task modification and task clearing. Thus, we specify a clearing operation at task 7 which clears task 3 and task 5. If either of these tasks are ongoing at the time, this clearing operation will cause the halting of these tasks. Since we no longer want the operator to monitor GCV 1, no signaling operation need be specified. In the same manner, we specify that the completion of task 8 causes the clearing of task 4 and task 6.

If, upon the completion of task 7, the operator is not performing either task 3 or task 5, then the clearing operation will have no effect. In this case, we must resort to task modification to insure that tasks 3 and 5 are not performed again.

For this situation, the information attribute packet representing GCV 1 will be awaiting scheduling of task 3 or task 5. Upon the completion of task 7, we modify task 3 into a task requiring no resources, taking no time, and having no branching. An information attribute packet waiting at task 3 will be immediately scheduled for the new task. Further, the new task will be completed in 0 time. Since no branching occurs from the task, the information attribute packet will be destroyed. Similarly, if we specify the modification of task 5 upon the completion of task 7, we can destroy the information attribute packet describing GCV 1 if it is awaiting scheduling at task 5. Likewise, for GCV 2, we modify both task 4 and task 6 into a task requiring no resources, taking no time to perform, and having no branching, upon the completion of task 8.

All of the above modifications involve the modification of a particular task into the same type of task. Thus, as illustrated in Figure 68 task 7 causes the clearing of task 3, the clearing of task 5, and the modification of both task 3 and task 5 into task 9 (a task with the characteristics described above). In the same manner, task 8 causes the clearing of task 4, the clearing of task 5, and the modification of both tasks 4 and 5 into task 9. The SAINT model illustrated in Figure 68 is a model of the same GCV system as previously described except that upon the arrival of each of the GCVs at its x-coordinate of 500,000 feet, the GCV will no longer be monitored by the operator. Since we have modeled the operator as not considering the GCV anymore, we do not have to regulate the value of the deviation to 0 nor reset the switch vector to 0 at tasks 7 and 8 for GCVs 1 and 2, respectively.

Summary

1. Task modification involves a substitution of the characteristics and output side of one task for those of another task upon the completion of a third task.

2. To represent task modifications, a dashed line is drawn from the modified task to the replacement task.

3. The number of the task whose completion causes the modification is placed inside a triangle alongside the dashed line.

4. Interchange modification allows for reflexive substitution of tasks.

5. Multiple replacement modification occurs when a task that has replaced another task is in turn replaced in the network.

6. Only interchange or multiple replacement modifications may be used in one SAINT model.

7. Task clearing involves the interruption (clearing) of a task in progress when another task is completed.

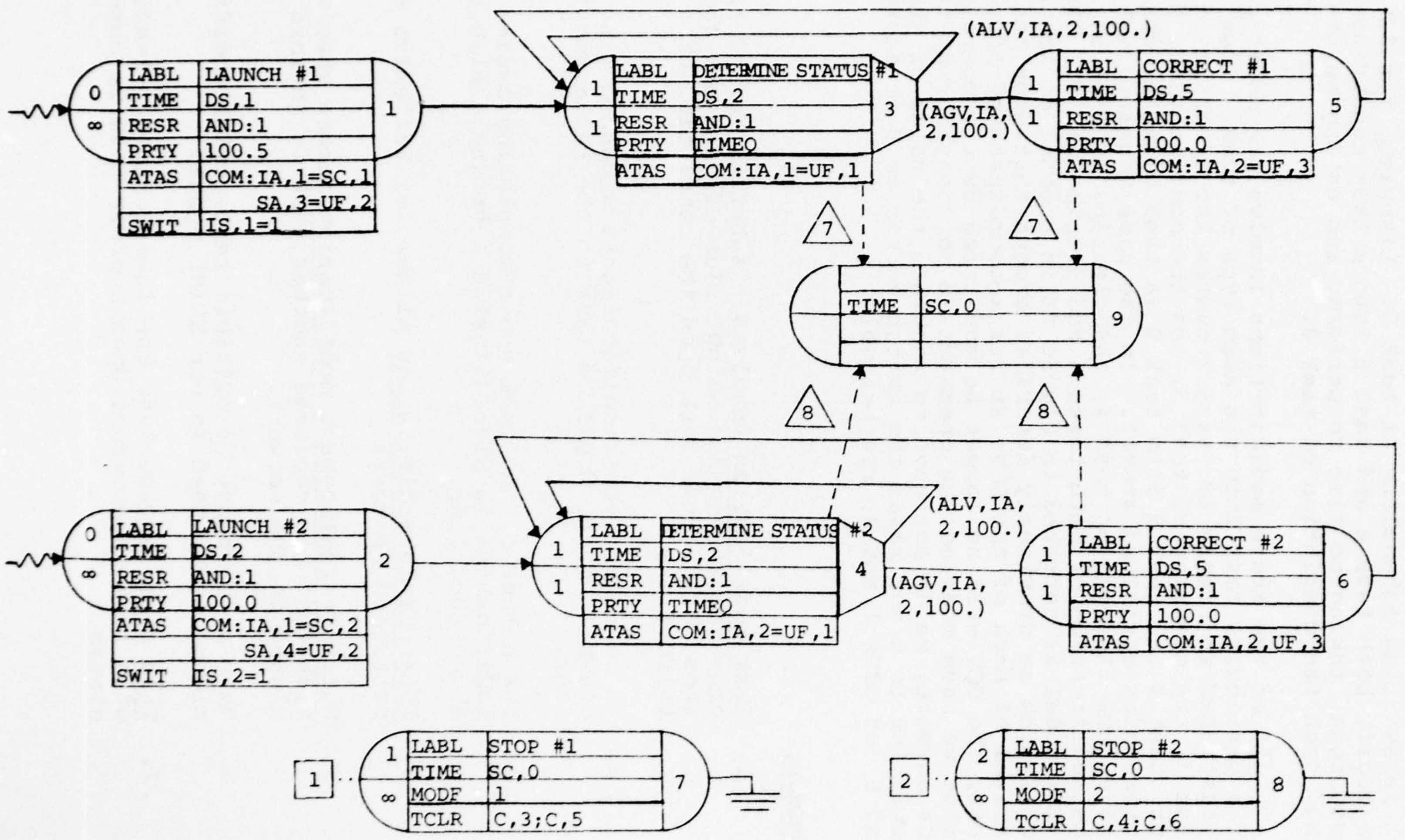


Figure 68. SAINT Model Illustrating Task Modification and Task Clearing.

8. If a clearing operation is performed, another task can be signaled (the transaction at the cleared task is sent to the signaled task and the predecessor requirements are reduced by one).

9. The task description code for task clearing (at the task causing the clearing) is TCLR.

10. The right-hand side of the TCLR row contains the specification of the task to be cleared and the task to be signaled.

11. To indicate a signaling operation, a circle containing the number of the task causing the signaling is connected to the task being signaled by a dotted line.

12. No clearing or signaling is performed if the task to be cleared is not ongoing at the time of the clearing operation.

SECTION XXII

RESOURCE CLEARING, DISTRIBUTION SET MODIFICATION, AND TASK COMPLETION PRECEDENCE

Resource Clearing

Previously, we have discussed the SAINT capability of task clearing. SAINT also provides a parallel capability for resources, i.e., resource clearing. Resource clearing involves halting a task in progress that is being performed by the specified resource. If the specified resource is cleared from a task, the task itself is cleared and all resources working on the task are set idle. However, if the resource to be cleared is idle at the time the clearing operation is to be performed, no clearing occurs. The signaling feature that is a part of task clearing may also be specified in the resource clearing operation.

Resource clearing is specified on the task symbol in the same manner as task clearing. The task description code for resource clearing is <u>RCLR</u>. In the right-hand portion of the RCLR row appears the letter C followed by the resource to be cleared and, if required, the letter S followed by the task to be signaled. To display signaling of a task, a circle is connected to the input side of the task to be signaled with a dotted line. Inside the circle appears the number of the task whose completion causes the signaling operation. Figure 69 displays the resource clearing specification at a task. At the completion of task 10, resource 2 will be cleared from either task 20 or 25 and task 30 will be signaled.

Distribution Set Modification

Distribution set modification is similar to task modification but involves the substitution of one distribution set for another based on some task completion. Once a distribution set modification has taken place, the task (or tasks) which previously used the original distribution set to generate performance times or attribute assignments will now use the substitute set. In addition, the same considerations regarding interchange and multiple replacement modification as discussed in the last section for task modification apply to distribution set modification.

Distribution set modification is described in the task description section of the task whose completion causes the modification. The task description code for distribution

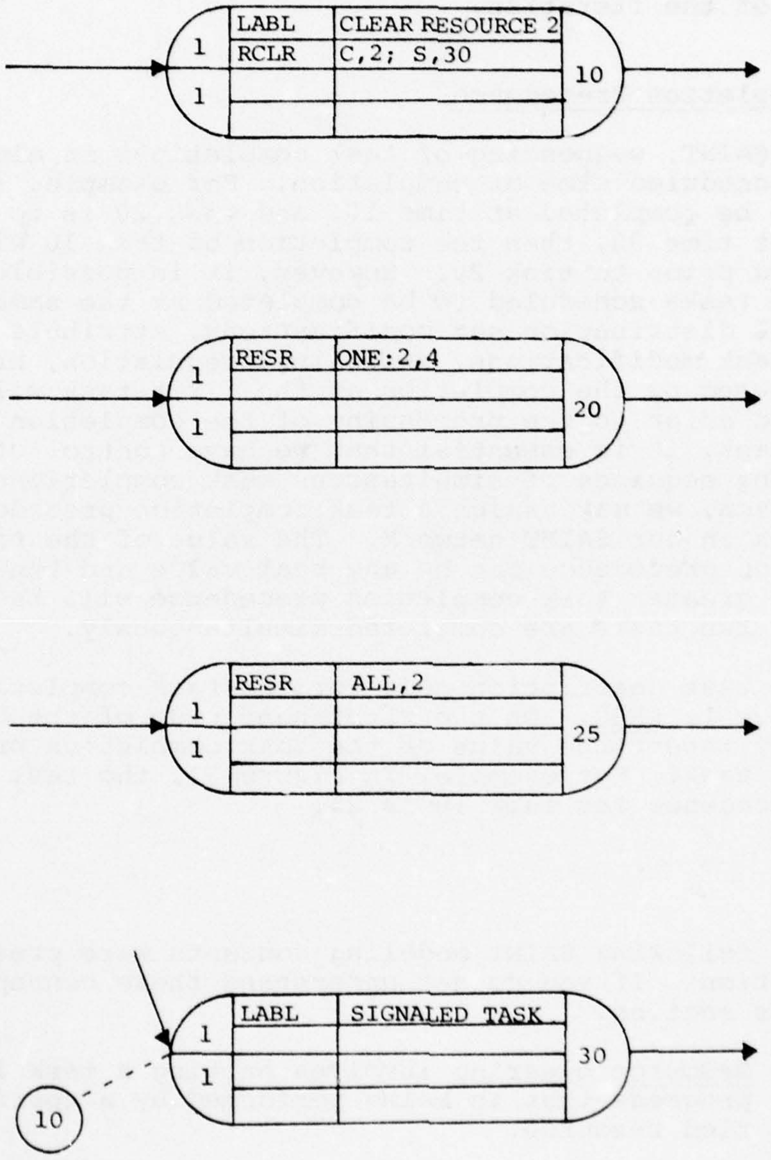


Figure 69. SAINT Representation of Resource Clearing.

set modification is DMOD. On the right-hand side of the DMOD row appears the distribution set number to be modified, the new distribution set to be used, and an arrow pointing from the old distribution set to the new. The distribution set modification specification is illustrated in Figure 70. Upon the completion of task 10, distribution set 3 will be replaced by distribution set 5 for the remainder of the iteration.

Task Completion Precedence

In SAINT, sequencing of task completions is always based on the scheduled time of completion. For example, if task 10 is to be completed at time 10, and task 20 is to be completed at time 30, then the completion of task 10 will be processed prior to task 20. However, it is possible to have two tasks scheduled to be completed at the same time. Since all distribution set modifications, attribute assignments, task modifications, switching, regulation, branching, etc., caused by the completion of the first task will be performed prior to the processing of the completion of the second task, it is essential that we have control of the processing sequence of simultaneous task completions. For this reason, we may assign a task completion precedence to each task in our SAINT network. The value of the task completion precedence can be any real value and the task with the greater task completion precedence will be processed first if two tasks are completed simultaneously.

The task description code for the task completion precedence is PREC. On the right-hand side of the PREC row we simply insert the value of the task completion precedence for this task. For example, in Figure 71, the task completion precedence for task 10 is 25.

Summary

The following SAINT modeling concepts were presented in this section. If you do not understand these concepts, reread this section.

1. Resource clearing involves halting a task in progress that is being performed by a specified resource.

2. Signaling can be performed after a resource clearing operation.

3. The task description code for resource clearing is RCLR.

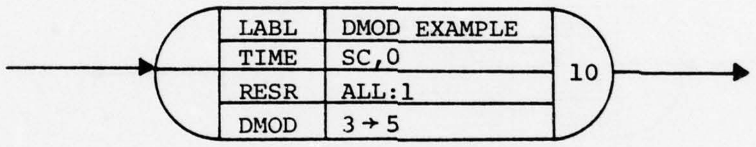


Figure 70. SAINT Representation of a Distribution Set Modification.

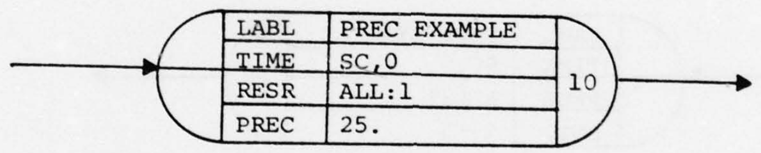
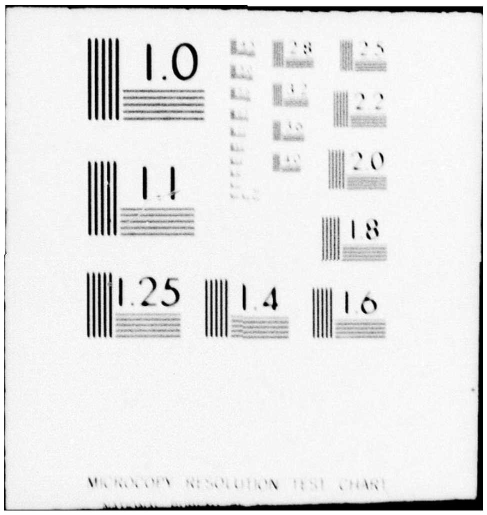


Figure 71. SAINT Representation of Task Completion Precedence.

AD-A058 671   PRITSKER AND ASSOCIATES INC WEST LAFAYETTE IND   F/G 9/2
SIMULATION USING SAINT: A USER-ORIENTED INSTRUCTION MANUAL.(U)
JUL 78   D B WORTMAN, S D DUKET, D J SEIFERT   F33615-76-C-5012
UNCLASSIFIED   AMRL-TR-77-61   NL

3 OF 3
AD
A058 671

END
DATE
FILMED
-11-78
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART

4. The right-hand side of the RCLR row contains the specification of the resource to be cleared and the task to be signaled.

5. To indicate a signaling operation, a circle containing the number of the task causing the signaling is connected to the task being signaled by a dotted line.

6. No clearing or signaling is performed if the resource to be cleared is not busy at the time of the clearing operation.

7. Distribution set modification involves the substitution of one distribution set for another based on some task completion.

8. Distribution set modification is described in the task description section of the task whose completion causes the modification.

9. The task description code for distribution set modification is DMOD.

10. The right-hand side of the DMOD row contains the distribution sets involved with an arrow pointing from the modified set to the replacement set.

11. The task completion precedence is used to order the processing of simultaneous task completions.

12. The task description code for the task completion precedence is PREC.

13. The right-hand side of the PREC row contains the value of the task completion precedence.

SECTION XXIII

CONCLUSION

As you have progressed from the beginning to the end of this instruction manual, you have been introduced to the extensive set of SAINT modeling capabilities and the procedures required to use them. In addition, you have been through the process of developing SAINT models of alternative configurations of the GCV system many times and have seen all of the SAINT modeling capabilities used in that context. As a result of this process, you should now be aware of the power and flexibility of SAINT as a tool for modeling complex systems.

By reading and studying this manual, you have gained a high degree of familiarity with the modeling concepts and procedures embodied in the SAINT technique and should now be able to construct SAINT models of the systems you wish to analyze. The next step is to learn the procedures that must be followed in order to convert SAINT models into data cards readable by the SAINT simulation program. These procedures are described in The SAINT User's Manual (1). It is only after studying the conversion procedures that you will be able to execute SAINT models.

Once you have executed a SAINT model, take the time to become familiar with the form and content of the output information SAINT provides. By doing so, you will gain a deeper appreciation for the wealth of system performance data that SAINT produces for you and will be better able to select and analyze SAINT outputs. Further, you will be prepared to learn additional procedures for analyzing SAINT output described in Analyzing SAINT Output Using SPSS (3).

Do not become discouraged if your initial SAINT modeling efforts prove to be difficult or frustrating. Modeling is an art that takes time and patience to master. By experimenting with the SAINT modeling capabilities and gaining confidence in your ability to use them, you will become an effective and efficient modeler capable of modeling a wide variety of complex systems using SAINT.

REFERENCES

1. Wortman, D.B., S.D. Duket, D.J. Seifert, R.L. Hann, and G.P. Chubb, The SAINT User's Manual, AMRL-TR-77-62, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio. A058724

2. Duket, S.D., D.B. Wortman, D.J. Seifert, R.L. Hann, and G.P. Chubb, Documentation for the SAINT Simulation Program, AMRL-TR-77-63, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio.

3. Duket, S.D., D.B. Wortman, D.J. Seifert, R.L. Hann, and G.P. Chubb, Analyzing SAINT Output Using SPSS, AMRL-TR-77-64, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio.

4. Kurke, M.I., "Operational Sequence Diagrams in System Design," Human Factors, 3, 1, March 1961, 66-73.

5. Wulff, J.J., J.N. Leonard, and A.F. Pixley, A Descriptive Model for Determining Optimal Human Performance in Systems, Volume I, National Aeronautics and Space Administration, Washington, D.C., January 1968.

6. Mitchell, M.B., R.L. Smith, and R.A. Westland, Techniques for Establishing Personnel Performance Standards (TEPPS). Procedural Guide (2nd ed.), Dunlap and Associates, Inc., Santa Monica, California, 1968.

7. Adams, J.A., and C.E. Weber, "Monte Carlo Model of Tracking Performance," Human Factors, 5, 1, February 1963, 81-102.

8. Pritsker, A.A.B., The GASP IV Simulation Language, New York: John Wiley & Sons, Inc., 1974.

BIBLIOGRAPHY

Askren, W.B., W.B. Campbell, D.J. Seifert, T.J. Hall, R.C. Johnson, and R.H. Sulzen, Feasibility of a Computer Simulation Method for Evaluating Human Effects on Nuclear Systems Safety, AFWL-TR-76-15, Air Force Weapons Laboratory, Kirtland Air Force Base, New Mexico, May 1976.

Chubb, G.P., "The Use of Monte Carlo Simulation to Reflect the Impact Human Factors Can Have on Systems Performance," Proceedings of the 1971 Winter Simulation Conference (Fifth Conference on the Applications of Simulation), held 8-10 December 1971, New York, sponsored by ACM/AIIE/IEEE/SHARE/SCI/TIMS, also identified by AMRL-TR-71-75.

Chubb, G.P., "Using Monte Carlo Simulation to Assess the Impact Radiation Induced Performance Degradation Can Have on Mission Success," presented at the Navy Symposium on Computer Simulation as Related to Manpower and Personnel Planning, Naval Personnel Research and Development Laboratories, Washington, D.C., July 1971.

Clayton, E.R., and L.J. Moore, "GERT vs. PERT," Journal of System Management, Vol. 23, February 1972, pp. 18-19.

Duket, S.D., D.B. Wortman, and D.J. Seifert, SAINT Simulation of a Remotely Piloted Vehicle/Drone Control Facility: Technical Documentation, AMRL-TR-75-119, AD A-029944, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio.

Emshoff, J.R., and R.L. Sisson, Design and Use of Computer Simulation Models, New York: The Macmillan Company, 1970.

Evans, G.W., II, G.F. Wallace, and G.L. Sutherland, Simulation Using Digital Computers, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1967.

Fishman, G.S., Concepts and Methods in Discrete Event Digital Simulation, New York: John Wiley and Sons, Inc., 1973.

Gordon, G., Systems Simulation, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1969.

Hann, R.L., and G.G. Kuperman, "SAINT Model of a Choice Reaction Time Paradigm," Proceedings of the 19th Annual Meeting of the Human Factors Society, AMRL-TR-75-25, AD A-027931, Dallas, October 1975.

Kuperman, G.G., and D.J. Seifert, "Development of a Computer Simulation Model for Evaluating DAIS Display Concepts," Proceedings of the 19th Annual Meeting of the Human Factors Society, Dallas, October 1975.

Maltas, K.L., and J.R. Buck, "Simulation of a Large Man/Machine Process Control System in the Steel Industry," Proceedings of the 19th Annual Meeting of the Human Factors Society, Dallas, October 1975.

Mirham, G.A., Simulation: Statistical Foundations and Methodology, New York: Academic Press, 1972.

Mize, J.J., and J.G. Cox, Essentials of Simulation, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1968.

Moder, J.J., and C.R. Phillips, Project Management with CPM and PERT, New York: Prinhold Publishing Corporation, 1964.

Moore, L.J., and E.R. Clayton, GERT Modeling and Simulation: Fundamentals and Applications, New York: Petrocelli/Charter, 1976.

Naylor, T.H., J.L. Balintfy, D.S. Burdick, and K. Chu, Computer Simulation Techniques, New York: John Wiley & Sons, Inc., 1965.

Pritsker, A.A.B., The GASP IV User's Manual, Pritsker & Associates, Inc., West Lafayette, Indiana, 1973.

Pritsker, A.A.B., The GERTE User's Manual, Pritsker & Associates, Inc., West Lafayette, Indiana, 1974.

Pritsker, A.A.B., The P-GERT User's Manual, Pritsker & Associates, Inc., West Lafayette, Indiana, 1974.

Pritsker, A.A.B., The Q-GERT User's Manual, Pritsker & Associates, Inc., West Lafayette, Indiana, 1974.

Pritsker, A.A.B., and P.J. Kiviat, Simulation with GASP II, A FORTRAN-Based Simulation Language, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1969.

Pritsker, A.A.B., and C.E. Sigal, The GERT IIIZ User's Manual, Pritsker & Associates, Inc., West Lafayette, Indiana, 1974.

Pritsker, A.A.B., D.B. Wortman, G.P. Chubb, and D.J. Seifert, "SAINT: Systems Analysis of Integrated Networks of Tasks," Proceedings of the Fifth Annual Pittsburgh Conference on Modeling and Simulation, Pittsburgh, Pa.: April 1974.

Pritsker, A.A.B., D.B. Wortman, C.S. Seum, G.P. Chubb, and D.J. Seifert, SAINT: Volume I. Systems Analysis of Integrated Networks of Tasks, AMRL-TR-73-126, AD A-014843, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio, April 1974.

Pritsker, A.A.B., and C.E. Sigal, The GERT IIIZ User's Manual, Pritsker & Associates, Inc., West Lafayette, Indiana, 1974.

Schmidt, J.W., and R.E. Taylor, Simulation and Analysis of Industrial Systems, Homewood, Ill.: Richard D. Irwin, Inc., 1970.

Schriber, T.J., Simulation Using GPSS, New York: John Wiley & Sons, Inc., 1974.

Seifert, D.J., and G.P. Chubb, Computer Models of Man-Machine Survivability/Vulnerability, AMRL-TR-72-69, AD 762528, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio, April 1973.

Shannon, R.E., Systems Simulation: The Art and Science, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1975.

Siegel, A.I., and J.J. Wolf, Man-Machine Simulation Models, New York: John Wiley & Sons, Inc., 1969.

Siegel, A.I., J.J. Wolf, and R.T. Sorenson, Techniques for Evaluating Operator Loading in Man-Machine Systems: Evaluation of a One or a Two Operator Evaluative Model Through a Controlled Laboratory Test, Applied Psychological Services, Inc., Wayne, Pa., 1962.

Siegel, A.I., J.J. Wolf, M.A. Fischl, W. Miehle, and G.P. Chubb, Modification of the Siegel-Wolf Operator Simulation Model for On-Line Experimentation, AMRL-TR-71-60, AD 737798, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio, June 1971.

Sigal, C.E., and A.A.B. Pritsker, "SMOOTH: A Combined Continuous-Discrete Network Simulation Language," SIMULATION, March 1974.

Sigal, C.E., "SMOOTH: A Combined Continuous-Discrete Network Simulation Language," Unpublished M.S.I.E. Thesis, Purdue University, West Lafayette, Indiana, August 1973.

Townsend, T., "GERT Networks with Item Differentiation Capabilities," Unpublished M.S.I.E. Thesis, Purdue University, West Lafayette, Indiana, August 1973.

Whitehouse, G.E., Systems Analysis and Design Using Network Techniques, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1973.

Wortman, D.B., S.D. Duket, and D.J. Seifert, "Simulation of a Remotely Piloted Vehicle/Drone Control Facility Using SAINT," Proceedings of the 1975 Summer Computer Simulation Conference, San Francisco, July 1975.

Wortman, D.B., S.D. Duket, and D.J. Seifert, New Developments in SAINT: The SAINT III Simulation Program, AMRL-TR-75-117, AD A-029894, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio, 1975.

Wortman, D.B., S.D. Duket, and D.J. Seifert, "SAINT Simulation of a Remotely Piloted Vehicle/Drone Control Facility," Proceedings of the 19th Annual Meeting of the Human Factors Society, Dallas, October 1975.

Wortman, D.B., A.A.B. Pritsker, C. Seum, D.J. Seifert, and G.P. Chubb, SAINT: Volume II. User's Manual, AMRL-TR-128, AD A-011586, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio, April 1974.

Wortman, D.B., C.E. Sigal, A.A.B. Pritsker, and D.J. Seifert, New SAINT Concepts and the SAINT II Simulation Program, AMRL-TR-74-119, AD A-014814, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio, April 1975.

Wortman, D.B., C.E. Sigal, A.A.B. Pritsker, and D.J. Seifert, SAINT II Documentation Manual, AMRL-TR-75-116, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio, April 1975.

Wortman, D.B., S.D. Duket, and D.J. Seifert, SAINT Simulation of a Remotely Piloted Vehicle/Drone Control Facility: Model Development and Analysis, AMRL-TR-75-118, AD A-031085, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio.

☆U.S.Government Printing Office: 1978 — 757-080/38